FACULTAD DE INFORMATICA
UNLP

TESINA DE LICENCIATURA

TITULO: Refactorings de los Modelos de Navegación y Presentación en Aplicaciones Web

AUTORES: Mazzei, Gustavo

DIRECTOR: Dr. Rossi, Gustavo

CODIRECTOR: Dra. Garrido, Alejandra

CARRERA: Licenciatura en Informática Plan 90

Resumen

Las aplicaciones Web evolucionan constantemente moldeándose al usuario, los procesos de desarrollo por excelencia son los procesos ágiles.

Estas metodologías iteran constantemente sobre las fases de desarrollo de la aplicación permitiendo que se introduzcan cambios en cada iteración. En este proceso la aplicación puede sufrir muchos tipos de modificaciones, algunas simples y otras más radicales.

Esta necesidad de cambio constante ha llevado a los desarrolladores a utilizar la técnica de refactoring, incrementando la calidad del diseño y del código mientras se preserva el comportamiento.

Indistintamente del proceso de desarrollo elegido, la mayoría de las metodologías de diseño para aplicaciones Web coinciden en tres modelos de diseño: de aplicación, de navegación y de presentación.

La técnica del refactoring, que en si fue concebida para reestructurar código, ahora se utiliza también a nivel de modelos.

En este trabajo me concentré en buscar las modificaciones más comunes que se realizan a aplicaciones Web y documentarlas de manera tal de formar un catálogo con los refactorings de modelo de navegación y presentación.

Con el creciente uso de los lenguajes orientados a objetos en el mercado y sobre todo en la tecnología Web, me pareció adecuado utilizar OOHDM para documentar los cambios en estos modelos.

Líneas de Investigación

Refactoring en aplicaciones Web.

Trabajos Realizados

Se provee un catalogo bien documentado de refactorings para el modelo de navegación y presentación de aplicaciones Web, mediante ejemplos resueltos sobre diagramas de navegación y presentación de OOHDM, mostrando su uso en aplicaciones del mundo real.

Conclusiones

Es posible mejorar de manera incremental y simple una aplicación Web mediante pequeños cambios y mejoras en sus modelos de navegación y presentación, sin cambiar los requerimientos básicos de la aplicación.

Se contribuye con un catalogo detallado de refactorings para el modelo de navegación y presentación, aplicados sobre aplicaciones del mundo real.

Trabajos Futuros

Desarrollo de un catalogo de refactorings de modelo de navegación que incluya los modelos diagramas de contexto de OOIIDM.

Mostrar los refactoring en implementaciones Web conocidas como struts, GWT, etc.

Desarrollar herramientas para automatizar los refactoring de los modelos de presentación y navegación.

Desarrollar un catalogo para refactorings de modelos de presentación para aplicaciones RIA.

Fecha de la presentación: Abril, 2009

Índice General

Índice de Figuras

Capítulo 1
Introducción

1.1 Motivación

En la actualidad las aplicaciones Web son aplicaciones que evolucionan constantemente moldeándose a la voluntad del usuario; por este motivo no es novedad que los procesos de desarrollo por excelencia sean los procesos ágiles.

Estas metodologías iteran constantemente sobre las fases de desarrollo de la aplicación permitiendo que se introduzcan cambios en cada iteración. En este proceso de cambio, la aplicación puede sufrir muchos tipos de modificaciones, algunas pueden ser ligeras y simples, mientras que otras pueden hacer cambios radicales dentro del sistema. Esto presenta a los diseñadores de las aplicaciones Web un gran desafío, ya que deben crear una aplicación que cambie constantemente y que mantenga la calidad, reusabilidad, testeabilidad y confiabilidad ante cada uno de estos cambios. Esta necesidad de cambio constante ha llevado a los desarrolladores a utilizar la técnica de refactoring para poder lidiar con estos cambios, incrementando la calidad del diseño y del código mientras se preserva el comportamiento.

Indistintamente del proceso de desarrollo que se utilice, las metodologías de diseño para aplicaciones Web son muy similares. La mayoría de ellas coincide en tres modelos de diseño: el modelo de la aplicación (o contenido), de navegación y de presentación [1, 2, 3].

Las metodologías de diseño y el uso constante de procesos ágiles de desarrollo, hacen que la técnica del refactoring, que en si fue concebida para reestructurar código, ahora se utilice también a nivel de modelos [4]. Los refactorings a nivel del modelo de aplicación han sido bien estudiados y han probado incrementar la reusabilidad y mantenibilidad de la aplicación mientras que preservan su comportamiento.

Luego de observar varios sitios populares considerados estado del arte en su rubro y que han tenido gran éxito en la red de redes, como Amazon, Barnes & Nobles, Google, Facebook, etc., se puede ver que para adaptarse a las necesidades del usuario han realizado mejoras constantes y evolucionado en el tiempo. Cabe destacar que la mayoría

de estas empresas invierten y disponen de una vasta cantidad de recursos para estudiar al usuario y su comportamiento en sus aplicaciones y de esta manera determinar la mejor manera de acercar el usuario a la aplicación.

Al analizar varios de estos sitios, en particular los de comercio electrónico, se ven determinados puntos de convergencia con respecto a diseño y funcionalidad, lo cual nos lleva a pensar que esa evolución natural devela cambios que pueden ser aplicados a sistemas de las mismas características para de poder mejorar su usabilidad y la percepción del usuario frente la aplicación.

En este trabajo me concentré en buscar las modificaciones más comunes que se realizan a aplicaciones Web y documentarlas de manera tal de formar un catálogo con los refactorings de modelo de navegación y presentación. El propósito de este catálogo es hacer un aporte al estudio de los refactoring de los modelos mencionados mostrando para cada uno de los refactorings estudiados, su motivación, mecanismo y aplicarlo en un sitio del mundo real, donde mostraremos su impacto y las mejoras que produce en el mismo.

Con el creciente uso de los lenguajes orientados a objetos en el mercado y sobre todo en la tecnología Web, me pareció adecuado utilizar OOHDM, para documentar los cambios en los modelos de las aplicaciones Web. OOHDM permite desarrollar una aplicación Web a través de una serie de iteraciones en las cuales en cada iteración se define un modelo, primero el de aplicación, luego navegación y finalmente el de presentación.

1.2 Organización de la Tesis

El trabajo está realizado en cuatro capítulos principales:

En el capítulo dos se hace una introducción sobre los trabajos relacionados, se introduce a las metodologías ágiles, la definición formal de refactoring, refactoring de modelos y se hace una introducción a los patrones de diseño Web, que constituyen la motivación de varios refactorings.

En el capítulo tres y cuatro, se definen formalmente los refactorings de modelo de navegación y presentación respectivamente, se describen los refactorings que se

estudiaron con su motivación, mecanismo y un ejemplo de algún sitio popular mostrando el antes y el después del refactoring.

En el capítulo cinco se explican las conclusiones de este trabajo y se sugieren trabajos futuros.

1.3 Contribuciones

Con este trabajo pretendo:

- Contribuir a la definición de refactorings para aplicaciones Web.
- Proveer un catalogo bien documentado de refactorings para el modelo de navegación de aplicaciones Web, mediante ejemplos resueltos sobre diagramas de navegación de OOHDM, mostrando su uso en aplicaciones del mundo real.
- Proveer un catalogo bien documentado de refactorings para el modelo de presentación de aplicaciones Web, mediante ejemplos resueltos sobre diagramas de presentación de OOHDM, mostrando su uso en aplicaciones del mundo real.

Capítulo 2
Trabajos Relacionados

Este capítulo presenta una revisión sobre los trabajos existentes en las áreas de metodologías ágiles, refactorings, patrones de diseño Web y la metodología OOHDM.

2.1 Metodologías Ágiles

Antes de hablar de refactoring debemos hacer una introducción sobre las metodologías o procesos de desarrollo ágiles.

En un proceso de desarrollo ágil se asume que el sistema crece con cada etapa, donde el usuario recibe las funcionalidades a medida que se realizan las distintas entregas del producto [5]. Dado que las funcionalidades se agrupan de acuerdo a distintas prioridades del cliente, puede ocurrir que al agregar nuevos requerimientos para la próxima iteración, se encuentren modificaciones o cambios que el cliente considera importantes de la iteración anterior, por lo que es necesario alterar las funcionalidades previas del sistema y agregar las nuevas funcionalidades.

Este modelo de proceso permite tiempos de entrega cortos, generalmente las iteraciones pueden durar entre dos semanas y un mes, lo cual permite introducir los cambios que el usuario solicita, la corrección de bugs o introducir nuevas tecnologías. Los requerimientos del usuario pueden introducirse en varios pasos en sucesivas iteraciones, mediante alteraciones parciales en cada entrega [6].

Seguir este proceso representa un desafío para los desarrolladores que se ven en la constante necesidad de modificar el diseño de la aplicación, no sólo para incorporar las nuevas funcionalidades que se requieren, sino también para modificar las ya existentes.

Dado que este es un proceso de continuo cambio, existen distintas formas de encarar estos problemas:

- Sobrediseñar el sistema, es decir, hacer uso intensivo de las técnicas de diseño, como por ejemplo patrones, en lugares del sistema donde no se sabe cuales serán los futuros requerimientos y de esta manera intentar preveer las posibles extensiones del sistema y evitar cambios sustanciales.

- Limitarse a cubrir los requerimientos existentes, es decir se programa sólo lo necesario para que el requerimiento sea cumplido, sin tener en cuenta cuales son las posibles modificaciones que se requerirán y sin pensar en un diseño extensible.
- Un punto intermedio entre los dos anteriores, es decir, realizar un diseño extensible sin llegar a sobrediseñar el sistema.

El primer caso es generalmente el que requiere más trabajo, dado que diseñar pensando en "lo que vendrá" implica tener un conocimiento vasto del dominio, de patrones y de técnicas de diseño. Esto generalmente trae complicaciones al momento de desarrollar dado que en el mercado es difícil encontrar un equipo que cumpla con estas condiciones. Si a este hecho le sumamos que hay que invertir mucho tiempo en diseñar y por lo general los tiempos de las iteraciones no superan el mes, el escenario puede ser muy complicado.

Si se posee un equipo altamente calificado que cumpla con las condiciones antes mencionadas, las complicaciones disminuyen ya que el sobrediseño deja de ser una complejidad y pasa a ser una ventaja al momento del desarrollo y generalmente paga con creces a la hora de realizar extensiones y modificaciones el tiempo invertido en el momento del inicio del desarrollo.

Cuando nos movemos usando el segundo enfoque, es decir, limitándonos a cubrir los requerimientos existentes, nos encontramos que luego de la primera iteración, las siguientes entregas se tornan muy difíciles ya que al no invertir tiempo en un diseño extensible, cualquier cambio que el cliente requiera implica un enorme esfuerzo para modificar el código que no está preparado para ser extendido. Lamentablemente este es un caso muy común en la industria debido a restricciones de tiempo y a que muchas empresas arman sus equipos de desarrollo con gente que proviene de diferentes ramas de la ingeniería en sistemas.

Con el tercer enfoque, si el tiempo de la iteración es suficiente podemos dedicar un tiempo al diseño antes de iniciar la codificación usando las buenas prácticas del diseño de sistemas. Es un punto intermedio entre el sobrediseño y el "no-diseño". En este enfoque, a diferencia del caso anterior, si se requiere un cambio que esté soportado por el

diseño, se podría realizar en forma relativamente fácil. No obstante si el cambio el muy radical, nos encontramos en el mismo dilema.

En estos momentos de cambios es donde la técnica del refactoring se ha vuelto popular, debido a que nos proporciona una manera elegante de poder realizar los cambios que no están soportados por el diseño y evita "emparchar" el diseño existente.

2.2 Refactoring

"Refactoring es el proceso de cambiar un sistema de software de tal manera que no altere el comportamiento externo del código y aún así mejora su estructura interna" [7].

"Es una manera de mejorar el diseño luego de que ha sido escrito" [9]. Se toma como entrada código que ha cambiado durante varias iteraciones, un diseño que se ha vuelto obsoleto como resultado de los sucesivos cambios, y como resultado obtenemos un buen diseño, extensible y mantenible.

Por definición, un refactoring nunca debe alterar el comportamiento externo del software, aunque no existe una definición formal de comportamiento [8], se propone que la condición de conservación sea que los métodos que se invocan antes y después del refactoring sean los mismos [9].

Esto permite a las técnicas ágiles desarrollar software en dos pasos iterativos. En el primer paso se desarrolla el comportamiento esperado y en el segundo se incrementa la calidad y la estructura del código sin cambiar el comportamiento original. [4]

Los distintos refactorings describen qué puede ser cambiado, cómo se deben hacer los cambios para no alterar la semántica y qué problemas esperar cuando se realizan estos cambios.

En general, no hay consenso acerca de la semántica asociada con la palabra "comportamiento" y como se prueba que el mismo no se altera luego de aplicar el refactoring. Esto se debe a que generalmente es muy difícil o muy caro probarlo [8].

Originalmente el refactoring se usaba para aplicar cambios a código orientado a objetos [10, 7], pero también más tarde se usó en lenguajes funcionales, imperativos y

lógicos (ej., Fortran [11], C[12] y Lisp[13]) como un método para reestructurar programas.

Siguiendo la dirección del refactoring de código, Ricca y Tonella han trabajado en reestructurar el código de aplicaciones Web [6]. Ellos definen diferentes categorías de reestructuración, como actualizaciones de sintaxis, mejoramiento interno de las páginas y creación dinámica de páginas, y especifican transformaciones se que aplican sobre HTML, PHP y JavaScript, que son algunos de los lenguajes utilizados para desarrollar aplicaciones Web. Ricca y Tonella implementaron algunas de sus reglas de reestructuración en la herramienta de reingeniería llamada DMS [14]. De manera similar, en su libro sobre refactoring de código HTML, Harold se enfoca en refactorings que permiten actualizar sitios a nuevos estándares Web como Extensible HTML (XHTML) y Cascading Style Sheet (CSS).

2.3 Refactoring de Modelos

Es ampliamente conocido que el software no es solamente código, sino más bien modelos que luego se derivan en código.

UML es el lenguaje de modelado de sistemas que se ha convertido en el estándar en la actualidad [15]. Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir el sistema, incluyendo aspectos conceptuales tales como procesos de negocio y funciones del sistema, aspectos concretos como diagramas de clases, expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables [15].

A pesar de que el refactoring originalmente fue pensado sobre código, el concepto de refactoring puede ser generalizado a una técnica que mejore la estructura del software y no solamente a la representación del código [4]. Se ha propuesto que esta técnica se aplique sobre los modelos de una aplicación, dando origen al concepto de refactoring de modelos. Pueden aparecer nuevos refactorings a nivel de modelo que no son aparentes desde el código [4].

La mayoría de los refactorings propuestos para modelos orientados a objetos han sido definidos en la estructura de clases de los sistemas y los cambios realizados por estos

refactoring son visibles en los diagramas de clases modelados en UML. Estos diagramas de clases son la estructura estática de un sistema, pero también se han realizado trabajos donde se aplica la técnica de refactoring a los diagramas que expresan el comportamiento dinámico como los procesos de negocio, diagramas de estado y diagramas de actividad [9].

En cada caso los refactorings son aplicados a diferentes artefactos y el comportamiento a preservar es definido para cada modelo particular, o no se define.

Con la aparición de las metodologías de diseño de aplicaciones Web (UWE [1], UWA [2], OOHDM [3], WebML [16]), que construyen la aplicación en diferentes modelos: aplicación, navegación y presentación, comenzaron a aparecer trabajos sobre refactorings aplicados a los diferentes modelos propuestos por estas metodologías [24], [17].

De las metodología nombradas anteriormente, sólo UWE[1], UWA[2] y OOHDM[3] son metodologías que usan el paradigma orientado a objetos para definir los modelos de la aplicación.

Los refactorings a nivel del modelo de aplicación han sido bien estudiados y han probado incrementar la reusabilidad y mantenibilidad de la aplicación mientras que preservan su comportamiento.

Dentro de algunos de los trabajos sobre refactorings de modelo de navegación podemos encontrar el de Cabot y Gomez [17], donde afirman que indistintamente del tamaño de la aplicación, algunos modelos de navegación pueden ser muy complejos y convertirse en enormes grafos difíciles de entender, lo cual hace que si su diseño no es extendible y claro, la calidad de la aplicación puede decaer de manera vertiginosa a medida que la aplicación crece. Es necesario aplicar refactorings para mejorar la calidad de esto modelos para de poder mantener la calidad y mejorar la mantenibilidad de la aplicación Web. [17]

El trabajo realizado por Cabot y Gomez, está realizado en WebML. En esta metodología el modelo de navegación consta de links y páginas que disparan operaciones sobre el modelo de la aplicación, que consiste en operaciones de lectura/actualización y borrado sobre las entidades que conforman el modelo. Los refactorings propuestos son sobre el modelo de navegación y preservan las operaciones sobre el modelo de la

aplicación. Ejemplos de sus refactorings son "Add Link", "Parallel Merge" (of pages) y "Head-Merge of Navigation Paths".

Una desventaja de este trabajo es que las operaciones definidas sobre esta metodología son básicas, y no poseen la complejidad que nos permite un modelo orientado a objetos.

En recientes artículos ([18, 24]), se ha descrito cómo las ideas detrás del refactoring pueden ser aplicadas a los modelos de una aplicación Web, generalizando el concepto de refactoring para incluir no solo mejoras en la estructura interna sino también para mejorar la usabilidad. Aquí extendemos esas ideas presentando un catalogo completo de refactorings a nivel de navegación y presentación de una aplicación Web

2.4 Patrones de diseño Web

Desde la aparición de los patrones de diseño, soluciones elegantes para que los expertos usaran en problemas recurrentes en el desarrollo de software, se han visto muchos catálogos de diseño, código, interfaces, e incluso de hipermedia y de aplicaciones Web.

Los patrones de hipermedia [23] dieron origen a los patrones Web en aplicaciones. Estos patrones Web, al igual que los patrones de diseño, toman en cuenta el conocimiento generado por la experiencia y por la resolución de problemas recurrentes, y derivan en una solución genérica que se puede instanciar en una aplicación específica según se requiera.

El lector puede encontrar catálogos de patrones Web en [19, 20, 21]. El catálogo más completo de patrones Web se puede encontrar en el trabajo de van Duyne et. al. [19].

Van Duyne en su libro expone un amplio catálogo de patrones de Web; su trabajo consiste en involucrar al usuario en el proceso de diseño de la aplicación, de la misma manera que lo realizan las metodologías ágiles.

Este catálogo lo que hace es centrarse en las necesidades del usuario y diseñar en base a sus necesidades, es decir utiliza el concepto de diseño centrado en el usuario.

Una de las contras que tiene este trabajo es que las soluciones a cada uno de estos patrones no utiliza ninguna metodología de desarrollo de aplicaciones Web, sino más bien sketchs con dibujos sobre la posible solución.

Otra desventaja es que no considera el diseño preexistente de la aplicación, por lo que su trabajo solamente es útil cuando se aplica a una aplicación que se construye desde el inicio. Por último, el hecho de que este trabajo no use una metodología de desarrollo Web, hace que los patrones estén aplicados sobre la interfaz y no a nivel del modelos de navegación.

2.5 OOHDM

OOHDM (Object Oriented Hypermedia Design Method) [3] es un proceso de diseño de aplicaciones Web, que consiste en tres etapas: la primera de ellas define el modelo de la aplicación o dominio, sobre este modelo se define en una segunda etapa un modelo de navegación y por último se define un modelo de presentación. La Figura 1, Figura 2 y Figura 3 muestran ejemplos de estos tres modelos.

Este trabajo se focaliza en refactorings sobre los modelos de navegación y presentación producidos por esta metodología.

A continuación se describirán brevemente los modelos presentados por esta metodología.

El **modelo de aplicación** (conceptual o de dominio) en OOHDM es un modelo orientado a objetos, modelado en UML con sus clases, sus relaciones y subsistemas.

Este modelo describe básicamente el contenido y comportamiento de nuestra aplicación, con las operaciones que nos permiten resolver la lógica del negocio. En este modelo se aplican todas las técnicas de desarrollo orientado a objetos que se conocen y los refactorings de modelos orientados a objetos. En el ejemplo de la La Figura 1 muestra el modelo conformado por tres clases: CD, Track y Artist. La clase CD representa un disco compacto y posee un conjunto de pistas (Track) y un artista (Artist). La clase CD posee un método que nos permite escuchar una introducción de cada Pista.

El **modelo de navegación** se construye sobre el modelo de aplicación. Puede ser considerado como una vista de este modelo. Estas vistas pueden ser diferentes de acuerdo a los distintos perfiles del usuario.

Para construir el modelo de navegación se deben identificar qué objetos serán navegables, qué atributos poseerán y cuáles serán las relaciones entre estos objetos con los objetos de la aplicación. Para realizar esta tarea se definen nodos y links.

Los nodos son representaciones de una o más clases del modelo de aplicación. Ésta correspondencia se describe a través de un lenguaje de consulta. Los nodos son las unidades que representan el contenido y operaciones de la aplicación que serán visitados por el usuario, contienen datos perceptibles (atributos), operaciones disponibles sobre los datos y anclas que permiten disparar los links. En la Figura 2 podemos ver que la clase de nodo CD es una representación de las clases de la aplicación CD, Artist y y Track y posee un método que dispara la operación del modelo para escuchar una pista del CD.

Los links representan las diferentes vías de navegación que poseerá el usuario para recorrer los nodos, estos reflejan las asociaciones del modelo de aplicación que tienen el propósito de ser explorados por el usuario final.

También se definen índices que son una forma particular de nodo compuesto para permitir navegación de uno a muchos. Cada entrada en un índice puede ser definido para contener algunos atributos del nodo destino y/o un ancla para navegar hacia él. Alternativamente, cada entrada puede ser un nodo completo.

También puede ser necesario combinar características de diferentes clases de la aplicación para describir la clase de un nodo. El mapeo entre los atributos de las clases de aplicación y los atributos de los nodos es especificado en el diagrama de clases de navegación con un lenguaje de consulta y el modelo de aplicación permanece en desconocimiento del mapeo [3]. No se describirá el lenguaje de consulta porque este trabajo no aplica cambios al mappeo.

El modelo de navegación es expresado en dos diagramas o esquemas: el diagrama de navegación y el diagrama de contexto.

Para este trabajo nos centraremos en los refactorings de modelo de navegación que aplican en el primero de los diagramas.

El **modelo de presentación** está definido como una capa sobre el modelo de navegación, mapeando los nodos a páginas y definiendo para cada página los objetos interfaz que mostrarán la información o facilitarán las acciones del usuario y navegación. En OOHDM, este modelo es también llamado Abstract Interface Design, y es descripto con una notación gráfica llamada Abstract Data View (ADV).

Un ADV en el diseño de la aplicación Web puede ser visto como un objeto de la interfaz, que tiene estado y comprende un conjunto de atributos, u otros objetos de interfaz, que definen su percepción y el conjunto de eventos que puede manejar, tales como eventos generados por el usuario. Algunos ejemplos son MouseClick, MouseDobleClick, etc. En la Figura 3 podemos ver el ADV asociado al nodo de navegación CD llamado CD_ADV que posee un conjunto de widgets que muestran los diferentes atributos del nodo asociado.

Los ADVs contienen diferentes tipos de widgets abstractos para mostrar atributos de nodos (ej. Campos Multimedia), disparar operaciones o permitir la navegación y también pueden contener otros ADVs.

El aspecto del comportamiento se especifica usando ADV-charts. Son una generalización de los diagramas de estado, pero a diferencia de ellos que solo permiten anidación estructural, los ADV-charts también soportan anidación estructural permitiendo estados internos dentro de ADVs y viceversa.

Los ADV-chart comprenden un conjunto de transiciones que gobiernan las transformaciones de interfaz. Cada transición especifica un evento que debe ser manejado, una precondición que debe ser satisfecha antes de disparar la acción y una post condición que indica el resultado de la interfaz luego de que la acción ha sido disparada [22].

Además tienen una variable reservada "perceptionContext" que se usa para indicar que las modificaciones en el espacio visual en un momento dado. Si agregamos objetos visibles al "perceptionContext" se visualizarán en la interfaz, y si se eliminan del mismo no serán perceptibles en la misma.

OOHDM usa ADVs para expresar: el layout de las páginas y las secciones de las páginas en términos de abstract widgets y agregaciones de ADVs; la conexión con el

modelo de navegación y las transformaciones de interfaz que ocurren como resultado de la interacción del usuario.

En términos de conexión con el modelo de navegación, una página del modelo de presentación puede mapear a uno o más nodos del modelo de navegación y de manera inversa, un nodo puede ser mapeado (no divido) entre varias páginas.

En la Figura 4 podemos ver una implementación del ADV, se muestra como vería el usuario final la representación abstracta del ADV.

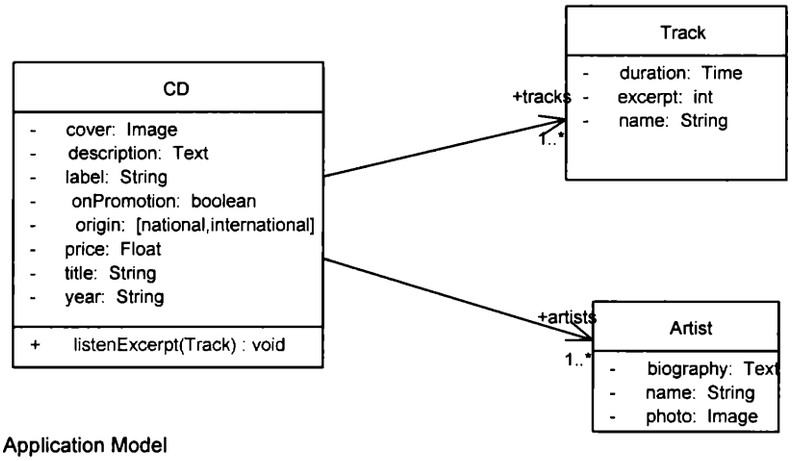


Figura 1 – Ejemplo de modelo de aplicación de OOHDM

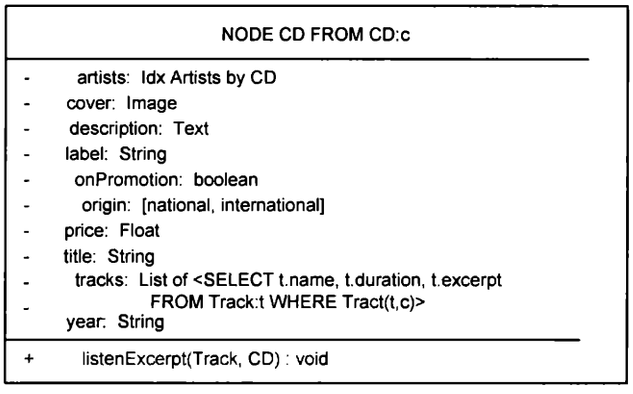


Figura 2 - Ejemplo de modelo de navegación de OOHDM

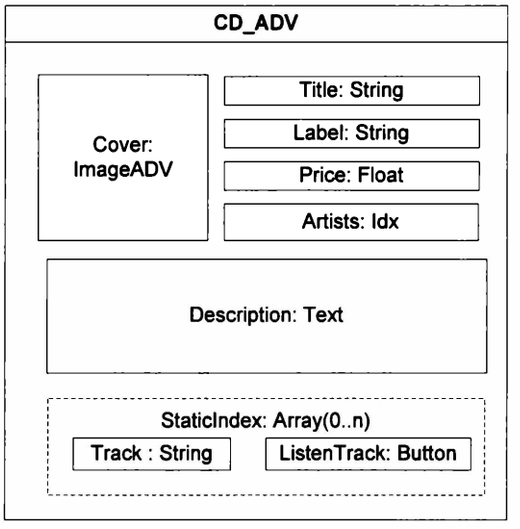


Figura 3 - Ejemplo de modelo de presentación de OOHDM

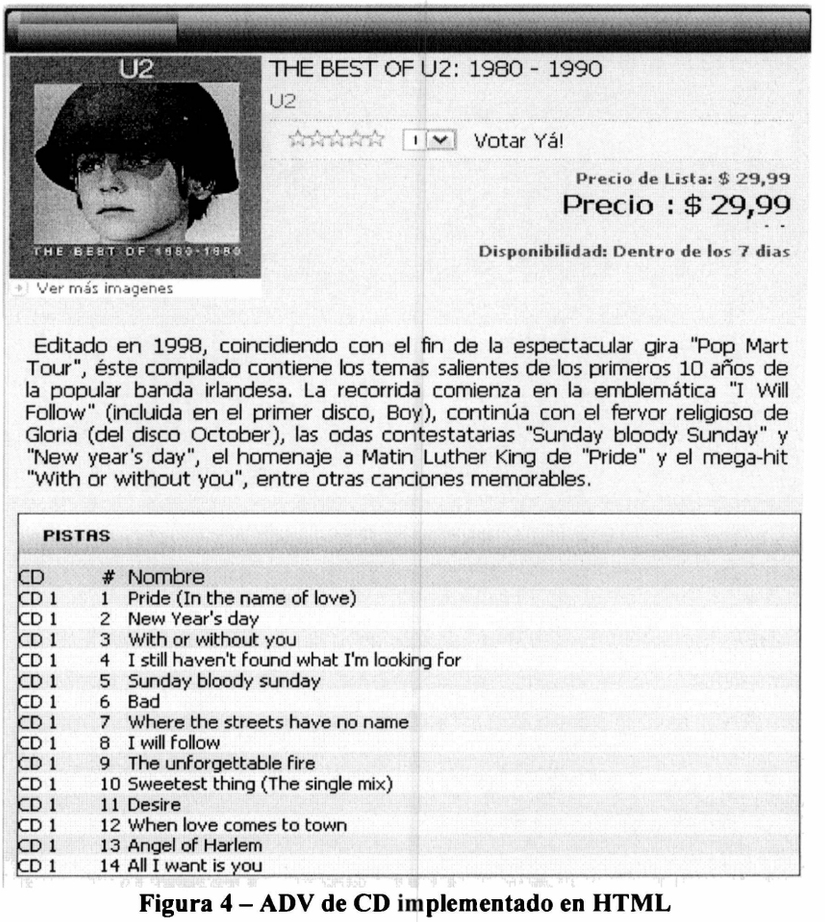


Figura 4 – ADV de CD implementado en HTML

Capítulo 3
Refactorings de Modelo de Navegación.

Según la metodología OOHDM, en los diagramas de navegación y de contexto se definen diversos elementos que nos permiten modelar la navegación. Nos limitaremos en esta sección a aquellos que se muestran sólo en los diagramas de navegación y dejaremos de lado los diagramas de contexto, por lo que los elementos afectados por estos cambios serán:

- Nodos.
- El contenido de los mismos (atributos, operaciones, anclas).
- Links entre nodos.
- Estructuras de acceso (índices).

Utilizaremos la definición dada por [24] para definir un refactoring de modelo de navegación.

Definición

Un refactoring del modelo de navegación producido según la metodología OOHDM afecta a los elementos del modelo de navegación. Estos refactorings realizan cambios sobre el modelo de navegación, conservando el comportamiento definido en el modelo de aplicación. Al definir comportamiento de la aplicación, nos referimos a los siguientes aspectos:

1. El conjunto de operaciones del modelo de la aplicación que son visibles desde el modelo de navegación y su semántica asociada.
2. El alcance de las operaciones a través de los links disponibles.

Los cambios aplicados por el refactoring deben preservar el comportamiento definido en el modelo de la aplicación y la navegabilidad del modelo de navegación. Asimismo no deben introducir información, relaciones u operaciones que no estén en el modelo de aplicación existente.

3.1 Caso de estudio

Antes de comenzar a describir nuestro catálogo definiremos un diagrama de navegación que será objeto de nuestros refactorings.

Nuestro caso de estudio será una tienda de cds on-line. El modelo de aplicación de esta tienda, que se puede ver en la Figura 5, consta de las clases CDShopApplication, CD, sus pistas (Track), artistas (Artist), clientes (Client), órdenes de Compra (Order), y los ítems de las ordenes de compra (OrderItem). El modelo de navegación correspondiente a este modelo de aplicación se muestra en la Figura 6. Este modelo consta de las siguientes clases de nodo:

- Order representa una orden de compra del modelo de aplicación.
- Item representa un OrderItem del modelo de aplicación.
- CD representa a un CD del modelo de aplicación.
- Artist representa un artista del modelo de aplicación.
- Bio qué representa la biografía de un artista del modelo de aplicación.

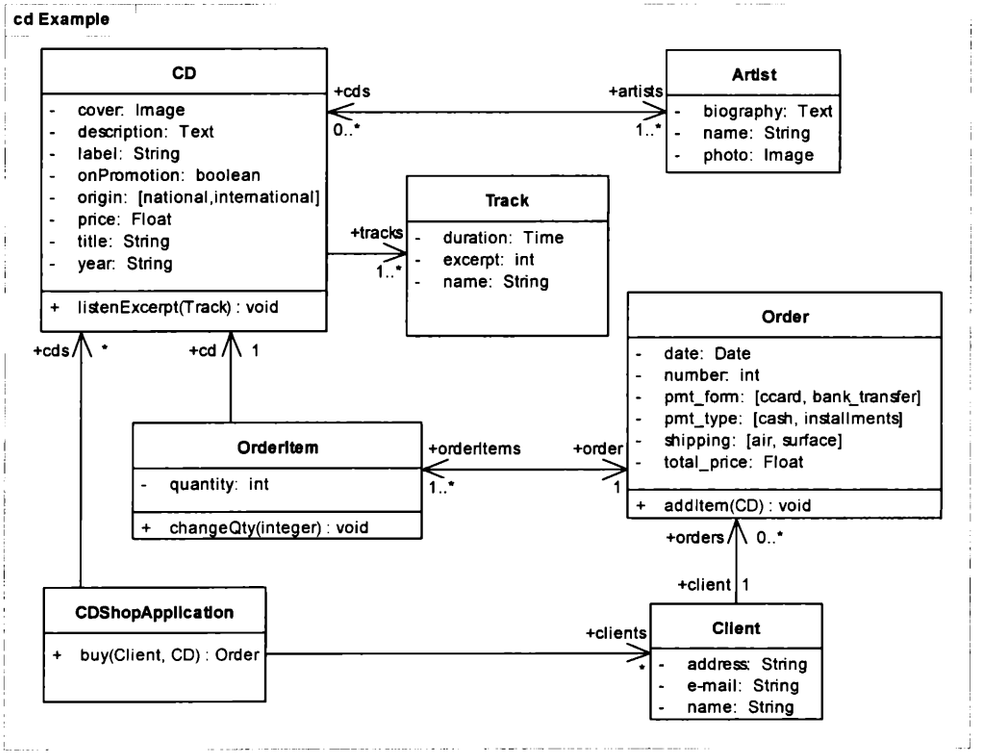


Figura 5 - Modelo de Aplicación – Caso de estudio

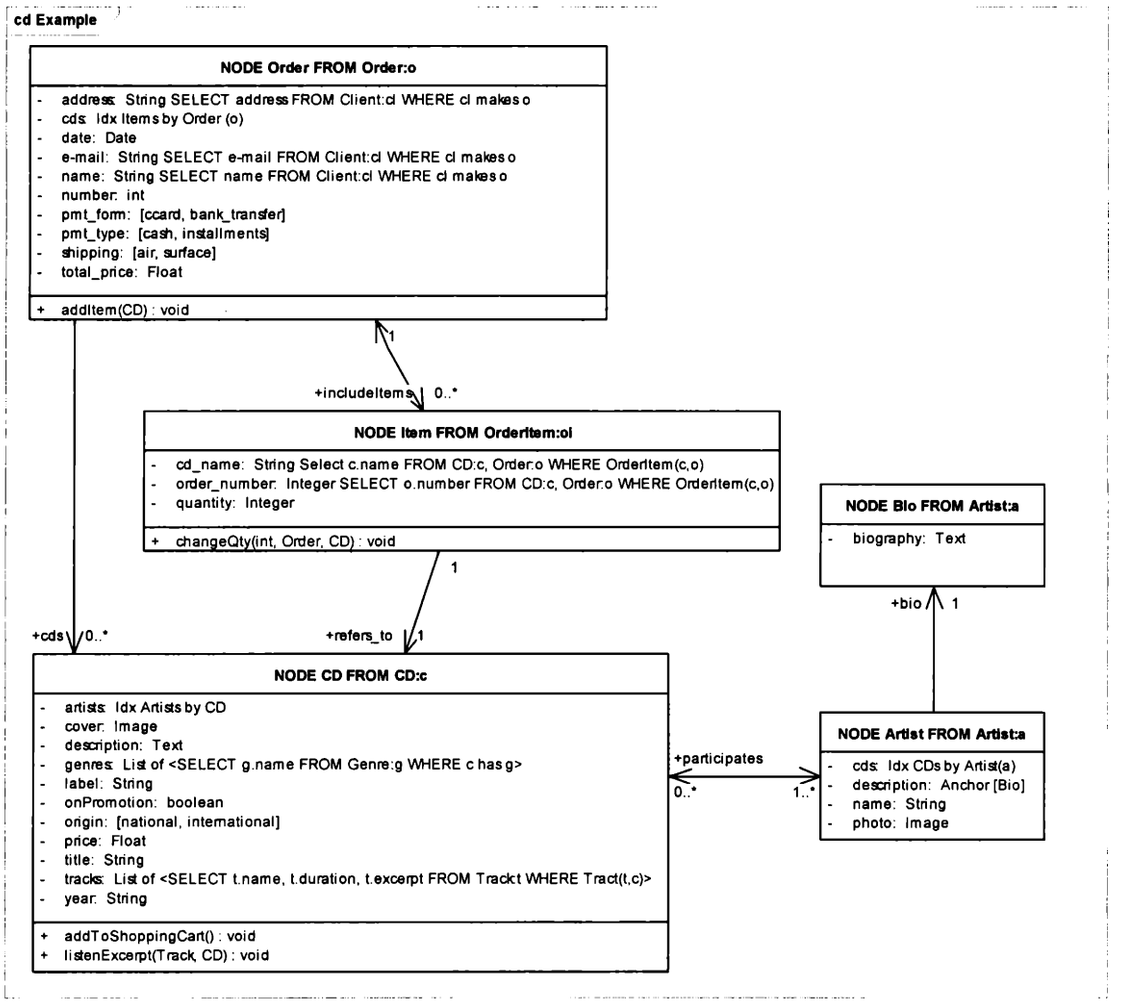


Figura 6 - Modelo de Navegación – Caso de estudio

Para ordenar el desarrollo de los refactorings de este catálogo, se seguirá la siguiente estructura:

- Motivación del refactoring.
- Mecanismo del refactoring y explicación en un diagrama genérico.
- Ejemplo concreto del refactoring sobre el caso de estudio.
- Refactorings relacionados.

3.2 Split Node Class

3.2.1 Motivación

La motivación de este refactoring es fragmentar un nodo que está repleto de información, saturado de links y funcionalidades. La idea es mover estos atributos y/o comportamiento del nodo en cuestión a un nodo nuevo.

También puede ser motivado por la introducción del patrón "Overview by detail" [20] o la introducción del patrón de modelo de aplicación "Extract Class" [7].

Dado que un nodo puede mapear una o más clases del modelo de aplicación, podemos decidir que el nodo actual puede ser muy abarcativo y decidimos crear un nodo nuevo para mapear una de las clases que antes estaba mapeada en conjunto con otra.

3.2.2 Mecanismo

Este refactoring de modelo de navegación se realiza en cuatro pasos:

1. Agregar una nueva clase de nodo vacía. El nombre de esta clase de nodo debe ser distinto a cualquiera de las ya existentes en el modelo de navegación.
2. Para cada atributo que se decida mover desde la clase del nodo original a la clase del nuevo nodo, usar el refactoring "Move Node Attribute" y agregar el atributo a la nueva clase de nodo. También se pueden copiar atributos que deberían estar en ambos nodos.
3. Para cada operación que se decida mover desde la clase del nodo original a la nueva clase de nodo, usar el refactoring "Move Node Operation". También se puede decidir copiar algunas operaciones.
4. Usar el refactoring "Add link" para agregar un link bi-direccional (o dos, uno de ida y otro de vuelta), entre la clase del nodo original y el nuevo nodo para permitir al usuario poder acceder la información original y su conjunto de operaciones.
5. Opcionalmente renombrar el nodo original para que refleje el nuevo contenido.

La Figura 7 muestra una descripción genérica de cómo se aplica este refactoring al modelo de navegación, con la vieja versión a la izquierda y la nueva a la derecha.

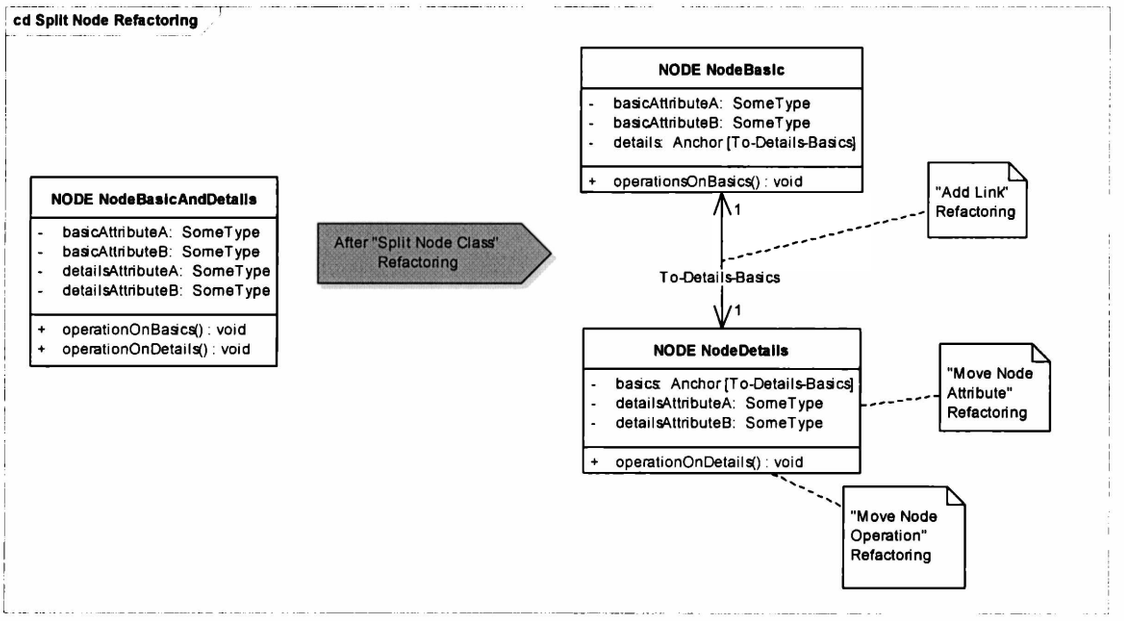


Figura 7 - Split Node Refactoring

3.2.3 Ejemplo

En nuestro ejemplo, podríamos decidir que la clase del nodo CD definida en el modelo de navegación es dividida en dos nuevas clases: CD_Basic y CD_Details). La Figura 8 muestra el diagrama de navegación antes y después del refactoring.

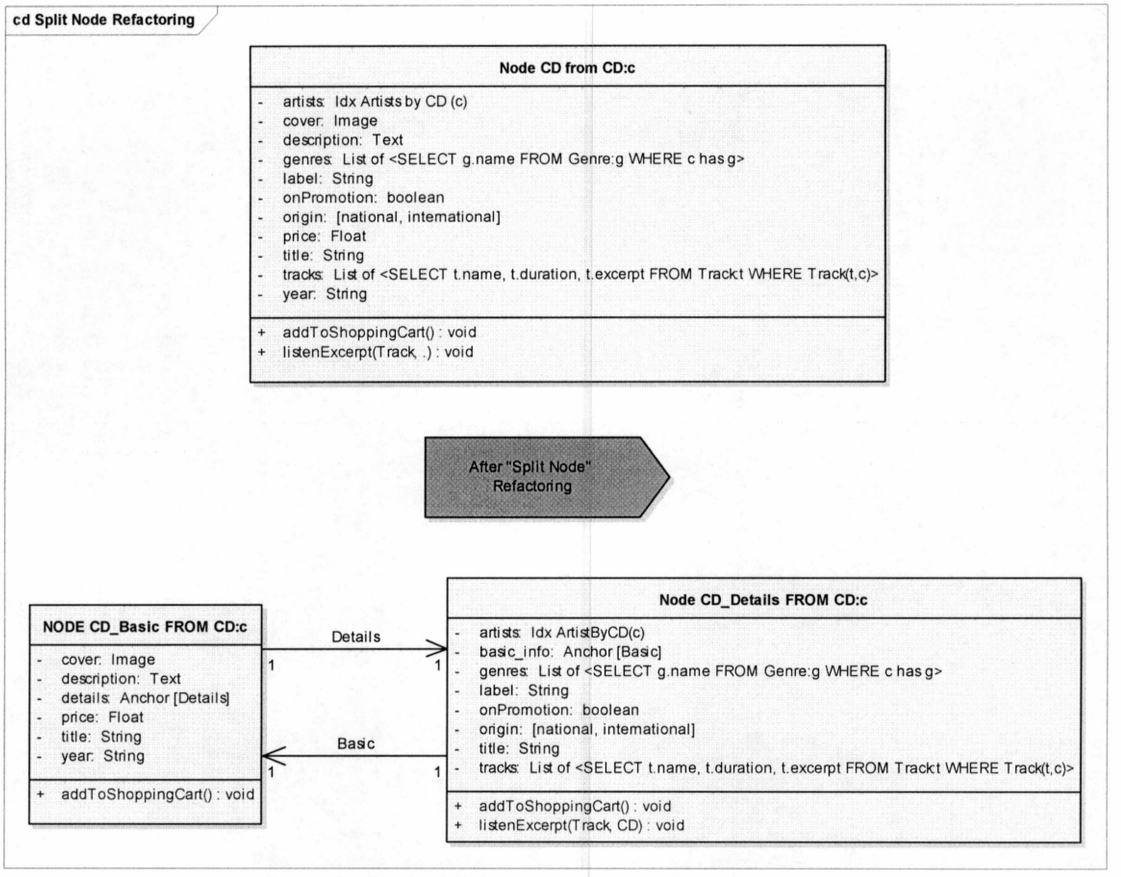


Figura 8 - Nodo CD luego del refactoring "Split Node"

3.2.4 Refactorings relacionados

Este refactoring es una composición de otros refactorings atómicos como: "Move Node Attribute", "Move Node Operation" y "Add Link".

Puede ser seguido por el refactoring "Split Page" o "Add Page Section" en el modelo de presentación.

3.3 Merge Node Classes

3.3.1 Motivación

La motivación de este refactoring, que es la inversa del anterior, consiste en unir dos nodos que presentan información sobre el mismo concepto de forma separada.

Este refactoring es aplicado cuando se determina que los usuarios que acceden a un determinado nodo, llamémoslo nodo A, siempre navegan a un nodo B, para realizar la tarea que desean, como por ejemplo, recuperar toda la información de un producto. Si el nodo de A no está demasiado cargado con información, puede ser conveniente unir los nodos A y B para acortar el camino de navegación, al menos para usuarios expertos.

Este refactoring también puede ser aplicado como consecuencia del refactoring "Inline Class" [7] en el modelo de la aplicación.

3.3.2 Mecanismo

Este refactoring se realiza de la siguiente manera:

1. Elegir la clase del nodo (A) al cual le será unido el otro (B).
2. Mover cada atributo de la clase del nodo B al A, usando el refactoring "Move Node Attribute". En el caso de que el atributo ya esté en la clase A, si el tipo es el mismo, ignorarlo, sino renombrar el atributo movido.
3. Mover cada operación de la clase del nodo B al A usando el refactoring "Move Node Operation".
4. Si hay un link desde B a A, borrar el link.
5. Para todos los otros links que salen de B, mover a A el origen de la flecha y la definición del ancla correspondiente.
6. Borrar el link desde A a B y su definición del ancla correspondiente (la existencia de este link está implícita por la motivación de este refactoring, dado que no queremos unir dos nodos no relacionados).
7. Para todos los otros links que llegan a B, cambiar su destino hacia A.

8. Considerar si hace falta renombrar la clase de nodo A en caso de que necesite incluir el nombre de B.
9. Aplicar el refactoring "Remove Unreacheable Node" en el B.

La Figura 9 muestra un diagrama genérico de una porción del modelo de clases de navegación afectado por este refactoring, con la vieja versión a la izquierda y la nueva a la derecha.

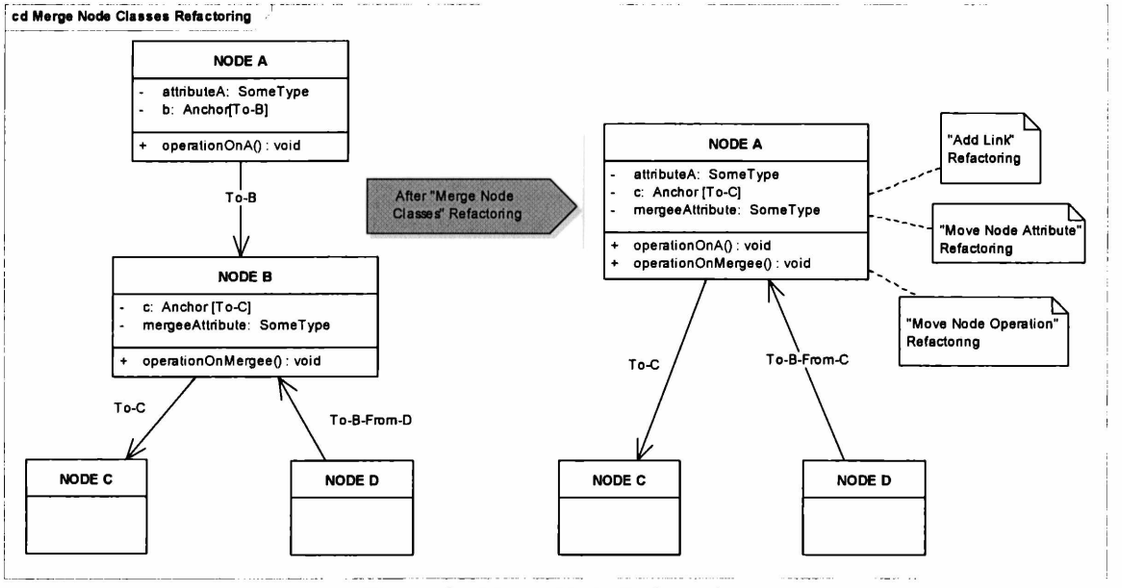


Figura 9 - Merge Node Refactoring

3.3.3 Ejemplo

En nuestro ejemplo, el nodo de la clase Bio puede ser unido a la clase Artist. La figura Figura 10 muestra el antes y el después del refactoring.

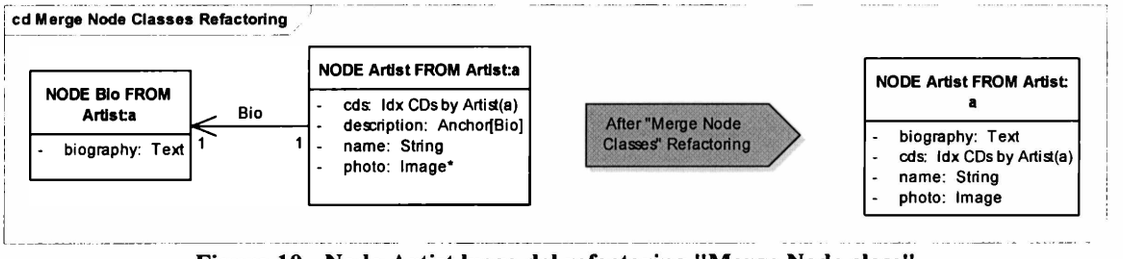


Figura 10 - Nodo Artist luego del refactoring "Merge Node class"

3.3.4 Refactorings relacionados

Este refactoring es una composición de refactorings más pequeños como "Move Node Attribute" y "Move Node Operation". También puede ser seguido por la aplicación del refactoring de presentación "Merge Page". Si el nodo absorbido (B) tenía su propia página en el modelo de presentación, hay dos opciones:

- Unir la página de B con la de A aplicando el refactoring "Merge Page"
- Si la página de A se vuelve muy sobrecargada, dejar la página de B pero actualizar el origen de su contenido.

3.4 Remove Unreachable Node

3.4.1 Motivación

La motivación de este refactoring de modelo de navegación es limpiar el modelo de navegación de nodos que no son alcanzables.

3.4.2 Mecanismo

El prerequisito para eliminar un nodo es que esté desconectado del resto de los nodos.

La Figura 11 muestra el refactoring para un nodo inalcanzable.

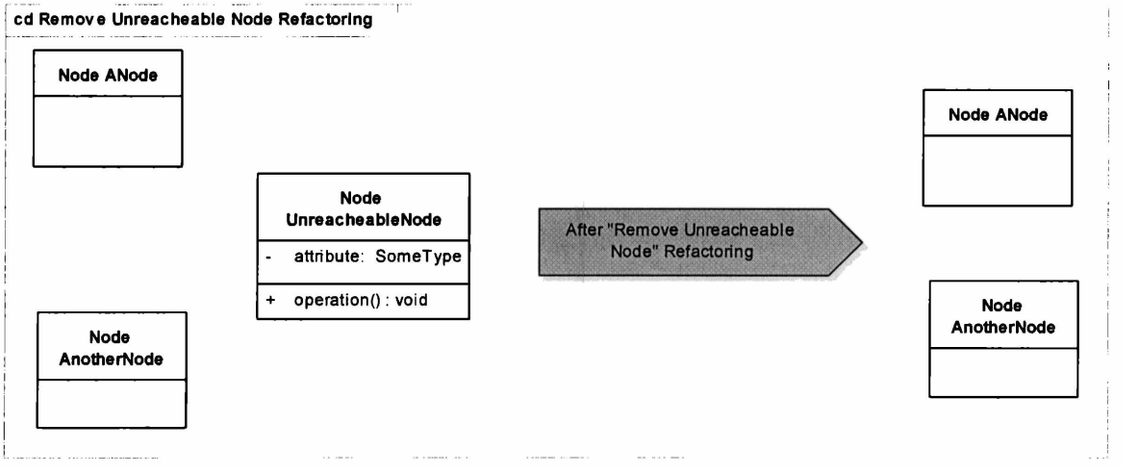


Figura 11 - Remove Unreacheable Node Refactoring

3.4.3 Ejemplo

En nuestro ejemplo de la Figura 10 el último paso de aplicar el refactoring de "Merge Node Class" es este refactoring y el resultado es el que se ve a la derecha de la figura.

3.5 Remove Redundant Node

3.5.1 Motivación

La motivación de este refactoring es limpiar el modelo de navegación de nodos que son redundantes. Esto se puede determinar a través de un análisis del tiempo de visita de cada nodo, donde se puede detectar que un usuario simplemente navega hacía otro nodo cuando navega el nodo redundante.

Otra motivación puede ser que el nodo representaba información y operaciones de una o más clases del modelo de aplicación que ahora desaparecieron.

3.5.2 Mecanismo

Hay dos casos posibles:

Que el nodo redundante este en un paso intermedio de la navegación o que sea un punto terminal.

Para el primer caso se procede de la siguiente manera:

1. Localizar el nodo redundante en el diagrama de clases del modelo de navegación (RedundantNode).
2. Identificar todos los links que salen de RedundantNode hacía otros nodos y usar el refactoring "Add link" para agregarselo a todos los nodos que tengan una navegación hacía él (excepto que el link ya exista).
3. Identificar todos los links que llegan a RedundantNode y eliminarlos, junto que sus correspondiente anclas.
4. Aplicar el refactoring "Remove Unreacheable Node" sobre RedundantNode.

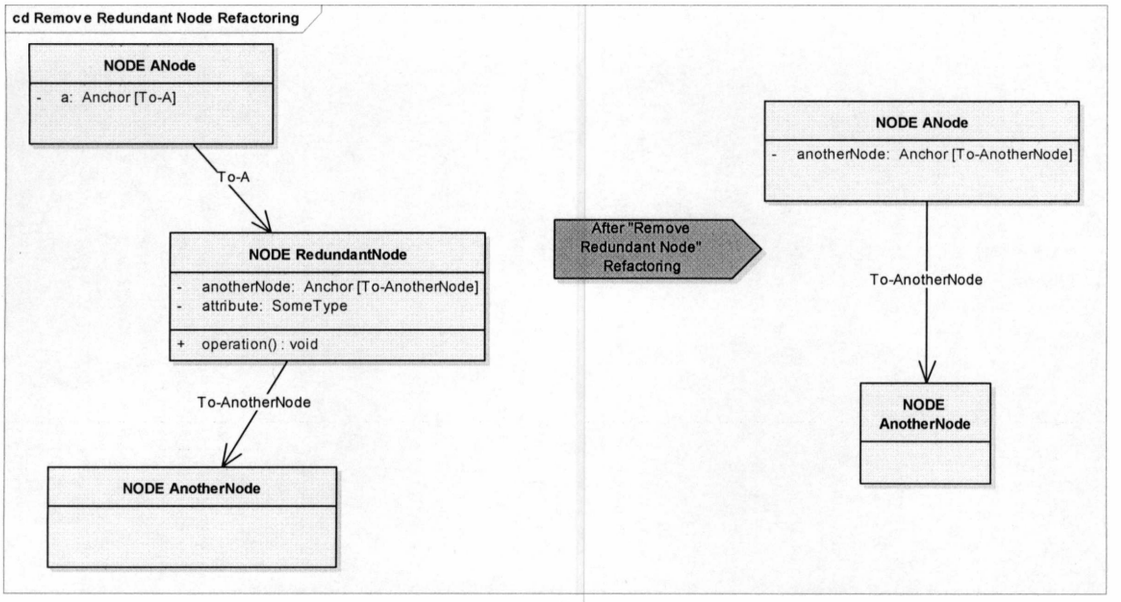


Figura 12- Remove Redundant Node Refactoring – Caso Nodo Intermedio

Para el segundo caso se procede de la siguiente manera:

1. Eliminar las anclas y los links que llegan al nodo redundante. (RedundantNode).
2. Aplicar el refactoring "Remove Unreacheable Node" al nodo redudante.

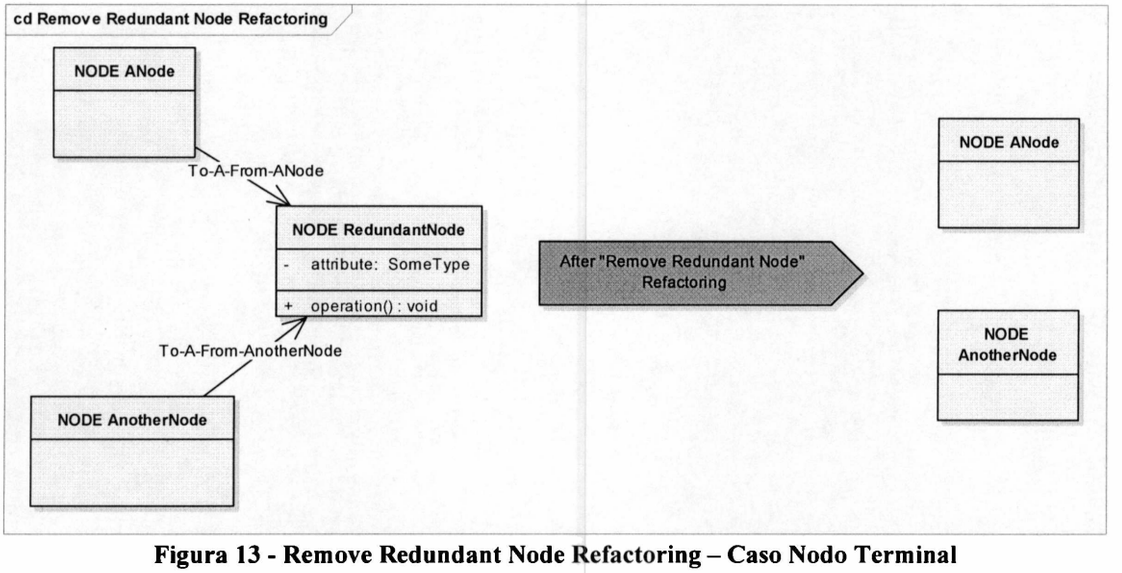


Figura 13 - Remove Redundant Node Refactoring – Caso Nodo Terminal

3.5.3 Ejemplo

En nuestro caso de estudio, podemos suponer que luego de un estudio de comportamiento del usuario en el sitio, nos damos cuenta que la biografía de los artistas no es visitada y por ende se decide eliminar el nodo. Este es el caso de un nodo terminal.

En la Figura 14 se puede ver la porción del diagrama afectada por el refactoring mostrando el antes y después del mismo.

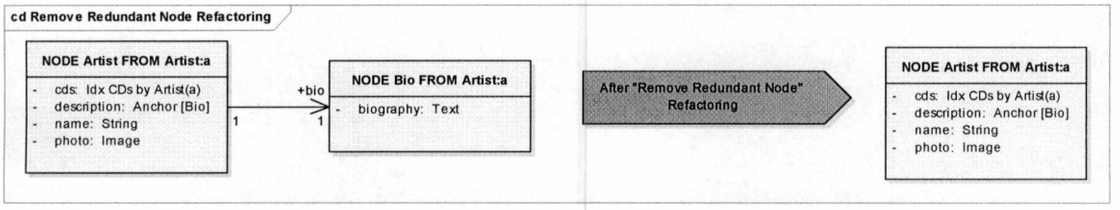


Figura 14 - Remove Redundant Node refactoring - Nodo Bio removido

Para el caso del nodo no terminal, podemos suponer que haciendo el mismo estudio de comportamiento, nos damos cuenta que el usuario siempre compra una sola copia de un Cd por Item de orden de compra, por lo que el nodo Item se vuelve obsoleto. La Figura 15 - Remove Redundant Node refactoring - Nodo Item removido.Figura 15 muestra el diagrama con el antes y el después del refactoring.

3.5.4 Refactorings Asociados

Luego de aplicar este refactoring sobre el modelo de navegación, se deberá eliminar en el modelo de presentación, el ADV que esté asociado o la página correspondiente.

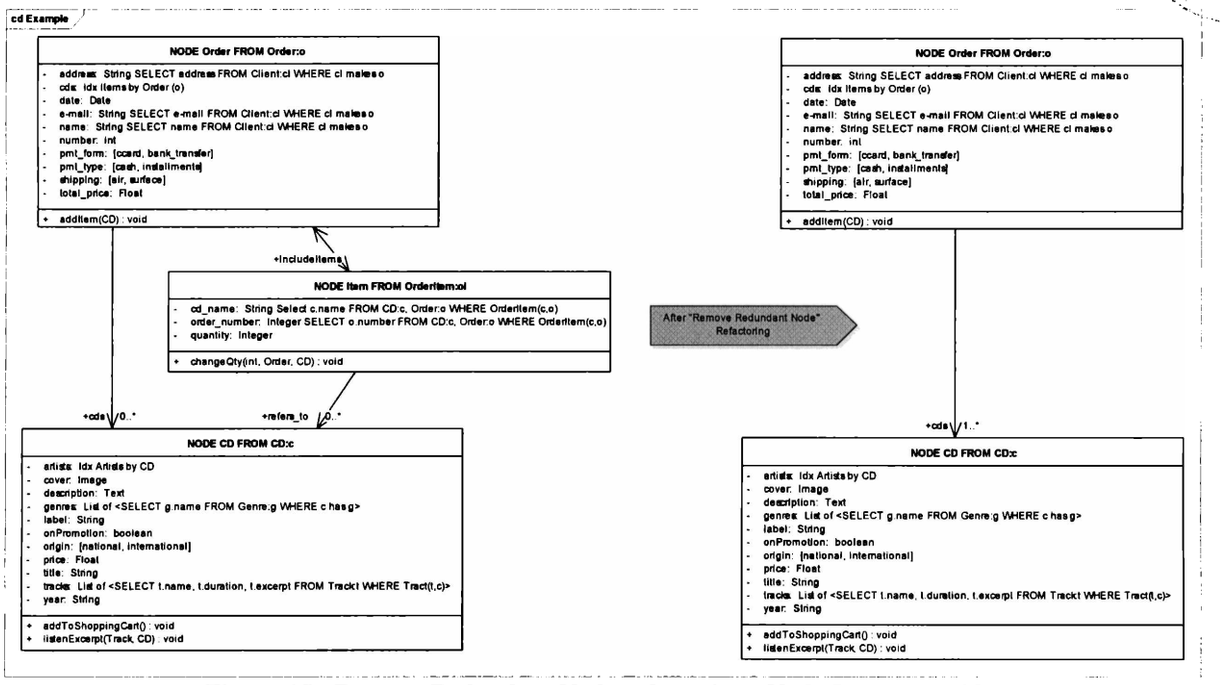


Figura 15 - Remove Redundant Node refactoring - Nodo Item removido.

3.6 Move Node Attribute

3.6.1 Motivación

Se determina que un atributo de una clase de un nodo origen necesita ser movido otra clase de un nodo destino.

Por el resultado de aplicar algún refactoring, como por ejemplo, "Split Node" se necesitan mover los atributos del nodo divido al nuevo nodo.

3.6.2 Mecanismo

Para mover un atributo de un nodo a otro se tiene que cumplir que: (1) la clase del nodo de origen y la clase del nodo destino deben existir en el diagrama de clases de navegación y (2) el atributo a mover no exista en la clase del nodo de destino, si es así, se procede de la siguiente manera:

1. Copiar el atributo en el nodo destino.
2. Eliminar el atributo del nodo origen.

La Figura 16 muestra un sketch de una porción del modelo de clases de navegación afectado por este refactoring, con la vieja versión a la izquierda y la nueva a la derecha.

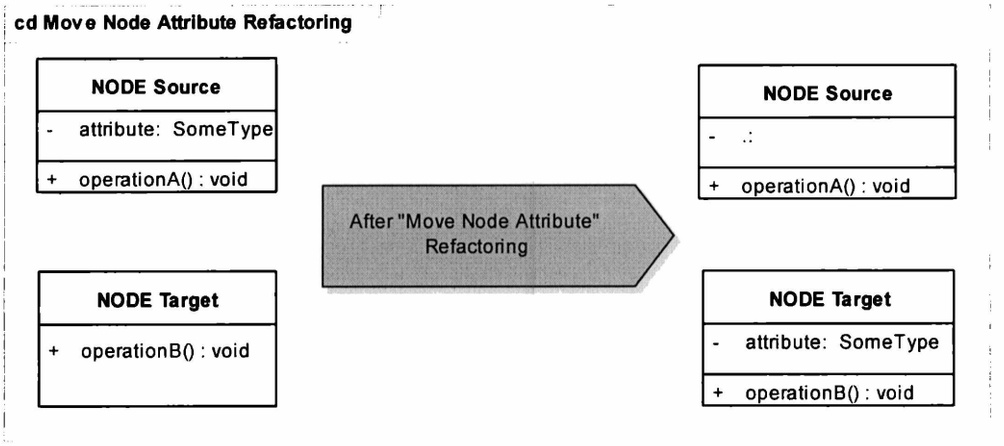


Figura 16 - Move Node Attribute Refactoring

3.6.3 Ejemplo

En Figura 8 vimos el resultado del refactoring "Split Node Class", y podemos ver cómo el atributo label del nodo CD es movido al nodo CD_Details.

3.7 Turn Attribute Into Link

3.7.1 Motivación

Al observar y estudiar el comportamiento del usuario en la aplicación Web, nos encontramos con que algunas veces existe la necesidad de transformar un atributo que se muestra en la pantalla en un camino de navegación hacía más detalles sobre lo que ese atributo representa.

Por ejemplo, durante el proceso de completar una transacción de negocios, algunas páginas Web pueden mostrar resultados intermedios o una revisión de la información recolectada hasta cierto punto (como un carro de compras). Dichas páginas Web deberían proveer al usuario una oportunidad de revisar la información asociada a los resultados intermedios a través de links a las páginas que muestran detalles sobre ellos.

También motiva este refactoring proveer "ayuda sensible al contexto" o "links embebidos".

3.7.2 Mecanismo

Seleccionar el atributo en el nodo de origen que mejor distinga al nodo de destino. En el diagrama de clases de navegación, realizar los siguientes dos pasos:

1. Usar el refactoring "Add Link" para agregar un nuevo link con nombre link-type-name desde el nodo de origen al destino, si el link no existe previamente.
2. En el nodo origen, reemplazar la definición del atributo por la definición de un ancla. La sintaxis de la definición del ancla en el modelo de navegación de OOHDM es la siguiente: anchor-name: [link-type-name]
3. Si el link entre el nodo origen y el destino ya existía, revisar si el ancla correspondiente sigue teniendo sentido o se podría borrar.

La Figura 17 muestra un sketch de una porción del modelo de clases de navegación afectado por este refactoring, con la vieja versión a la izquierda y la nueva a la derecha.

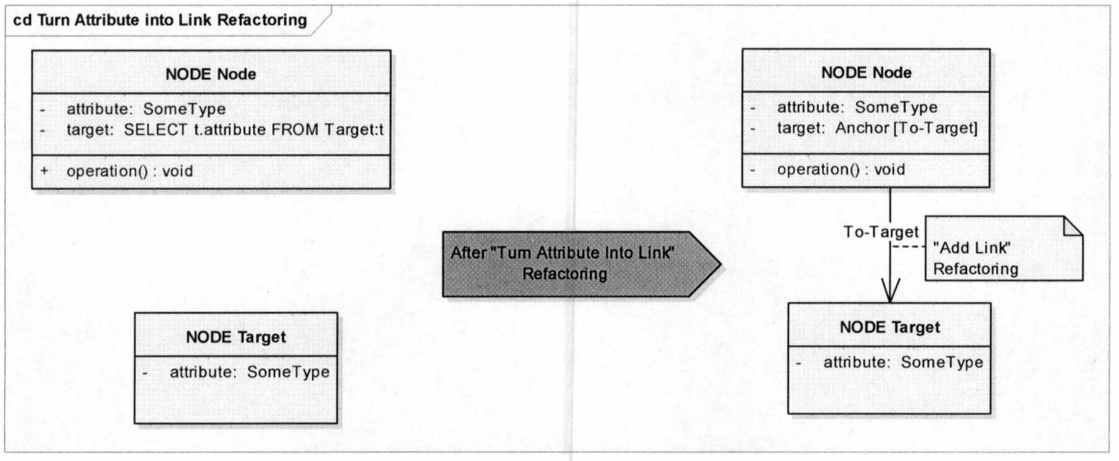


Figura 17 - Turn Attribute Into Link Refactoring

3.7.3 Ejemplo

A veces en los sitios de comercio electrónico, nos encontramos que durante el proceso de checkout, el usuario puede visualizar los ítems que posee en el carrito de

compras. En algunos de estos sitios, el usuario sólo puede visualizar el nombre de los ítems que ha comprado junto con la cantidad solicitada, pero es incapaz de entrar a los detalles de los ítems en el carrito.

Al aplicar el refactoring transformamos el atributo cd_name en el nodo del Item, para que se transforme en un ancla. En la Figura 18 se muestra el antes y el después de este refactoring. El attributo cd_name en los diagramas anteriores poseía la query que recuperaba el nombre del CD del modelo de aplicación, dado que esa query es simplemente ilustrativa, en el modelo es simplemente un string que se muestra con esos datos.

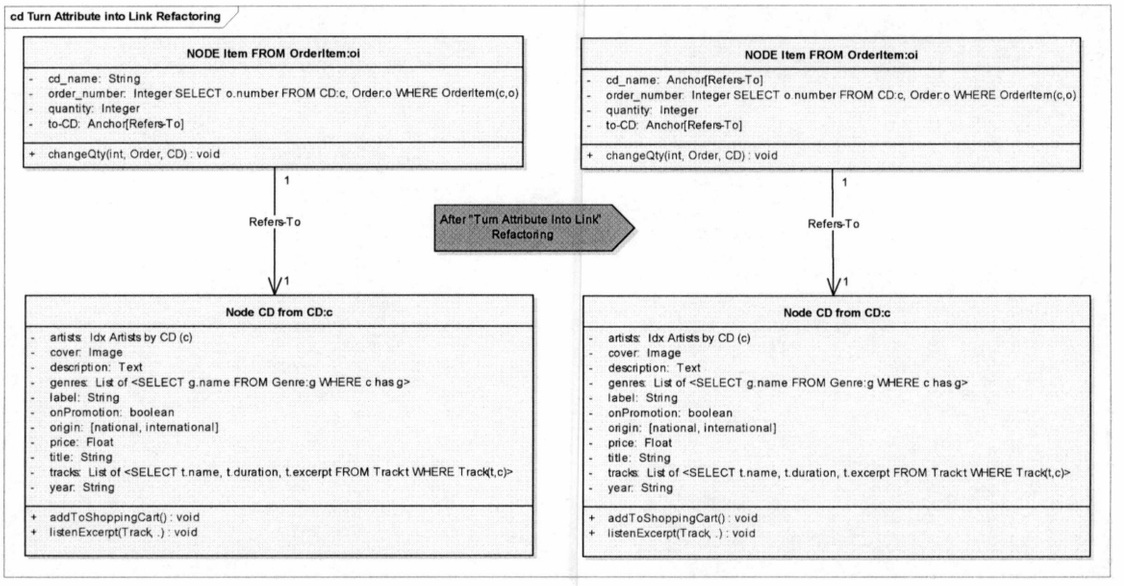


Figura 18 - Turn Attribute into Link refactoring - Ejemplo

3.7.4 Refactorings asociados

En el modelo de presentación, el refactoring "Add Interface Anchor" se necesita para mapear el ancla que se introduce en la definición del nodo.

3.8 Add Node Operation

3.8.1 Motivación

La motivación de este nodo es acercar una operación del modelo de aplicación a un nodo del modelo de navegación. Esto puede ser porque se agrega una nueva operación en el modelo de aplicación y queremos que esté disponible en el modelo de navegación, o porque se determina que se quiere acercar una operación al usuario durante la navegación, es decir que el usuario no deba navegar hasta determinado nodo para ejecutar la operación en cuestión, esto generalmente puede ser realizado en los procesos de check-out.

Las operaciones también se pueden agregar a cada entrada en un índice, así el usuario no necesita navegar el nodo describiendo la entrada para operar con el mismo.

3.8.2 Mecanismo

La precondición para agregar una operación a una clase del nodo en el modelo de navegación es que la operación ya esté disponible en el modelo de la aplicación.

Si la precondición se cumple, la operación se puede agregar al nodo en la parte inferior de la definición del nodo.

La Figura 19 muestra un sketch de una porción del modelo de clases de navegación afectado por este refactoring, con la vieja versión a la izquierda y la nueva a la derecha.

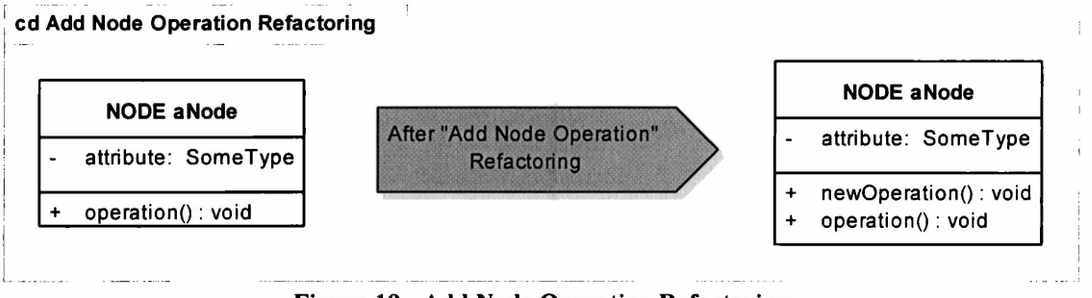


Figura 19 - Add Node Operation Refactoring

3.8.3 Ejemplo

En la tienda Amazon.com, el proceso de check-out ha evolucionado con el tiempo para permitir un incremento de la velocidad. Cuando se agrega un item al carrito de compras, el carro posee un botón extra que dice "Comprar con un click", y al presionarlo le permite al usuario ingresar al sitio, y recuperar toda la información de una orden previa, evitándole al usuario tener que volver a ingresar la información otra vez cada vez que realiza una compra.

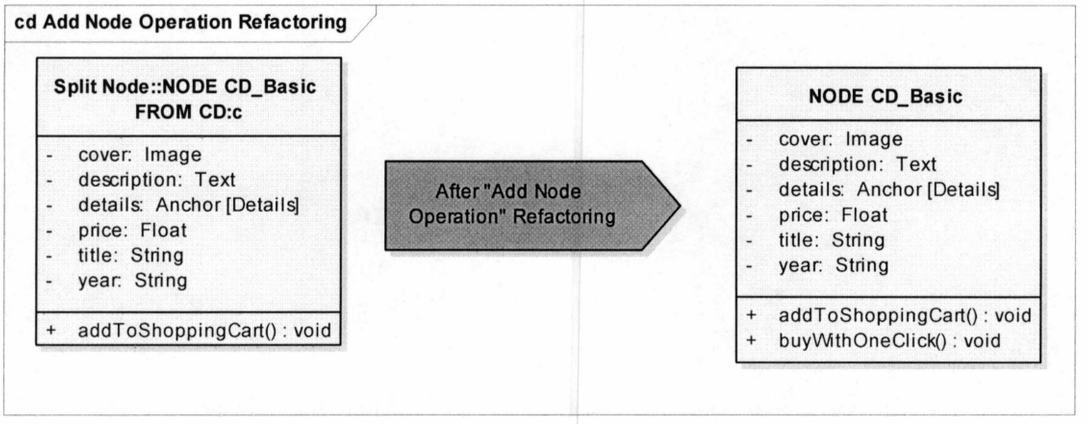


Figura 20 - CD_Basic luego del refactoring "Add Node Operation"

3.8.4 Refactorings relacionados

Agregar una operación el modelo de navegación requiere que el modelo de presentación sea extendido con un widget para disparar la operación. Esto requiere que el ADV para la página correspondiente sea reestructurado, por ejemplo, usando el refactoring "Add Page Section" para agrupar las operaciones relacionadas. Adicionalmente, si la operación puede tardar mucho tiempo en completarse se puede aplicar el refactoring "Add Processing Page".

3.9 Move Node Operation

3.9.1 Motivación

Luego de estudiar el comportamiento de los usuarios en un sitio Web, se puede determinar que el usuario debe navegar por varios nodos hasta llegar a realizar su operación. Luego se puede decidir mover la operación de un nodo a otro para acortar el camino de navegación.

También puede ser motivación el resultado de aplicar el refactoring "Split Node Class" o "Merge Node Classes".

3.9.2 Mecanismo

Sean dos nodos Source y Target, si las siguientes precondiciones se cumplen: (1) ambos existen en el diagrama de clases de nodos y (2) la signatura de la operación a mover no se encuentra ya definida en la clase Target, se procede de la siguiente manera:

1. Se agrega en la definición de la clase del nodo Target la operación del nodo Source.
2. Se elimina la operación de la clase del nodo Source.

La Figura 21 muestra un sketch de una porción del modelo de clases de navegación afectado por este refactoring, con la vieja versión a la izquierda y la nueva a la derecha.

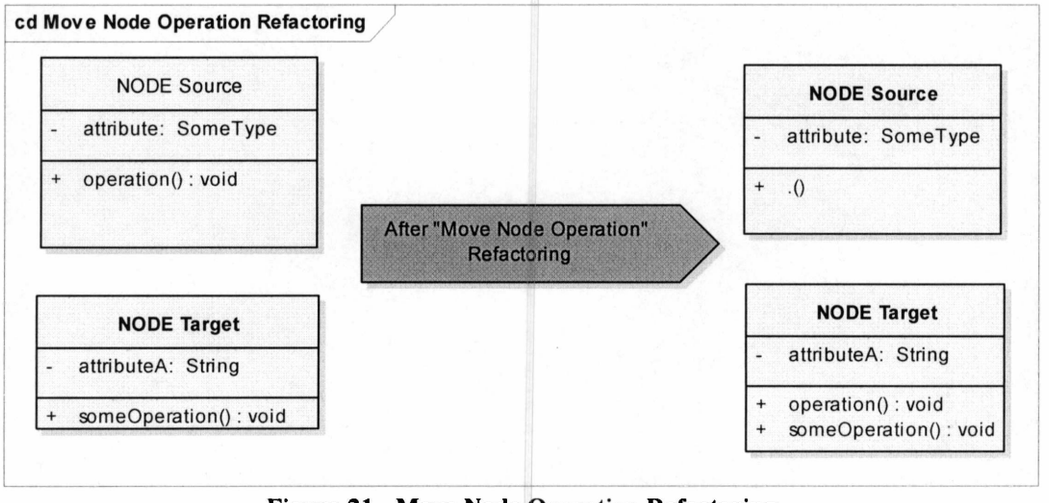


Figura 21 - Move Node Operation Refactoring

3.9.3 Ejemplo

En nuestro ejemplo del refactoring "Split Node Class" de la Figura 8 podemos ver como los atributos del nodo original, son movidos por este refactoring al nodo CD_Details.

3.9.4 Refactorings Relacionados

Este refactoring esta relacionado con los refactorings "Merge Node Class", "Split Node Class" y como consecuencia en los modelos de navegación con "Move Widget".

3.10 Add Link

3.10.1 Motivación

La motivación de este refactoring puede ser la de haber previamente aplicado el refactoring "Split Node Class", para permitir al usuario navegar entre los dos nodos resultantes.

Este refactoring también se aplica cuando es necesario proveer al usuario un camino nuevo de navegación –acortar el camino– entre nodos distintos que ya son alcanzables desde otro nodo.

Otra motivación para este refactoring puede ser permitir al usuario navegar hacia un nuevo nodo que ha sido agregado recientemente en el diagrama de navegación.

3.10.2 Mecanismo

Un nuevo link llamado To-Target entre dos nodos Source y Target puede ser agregado sólo si las siguientes precondiciones se cumplen:

(1) Source y Target existen como clases de nodos en el modelo de navegación .

(2) To-Target no está siendo usado en el modelo de navegación.

(3) No existe un link entre Source y Target con la misma semántica del que se quiere agregar, lo cual crearía un link redundante. Si estas precondiciones se cumplen, un nuevo link se agrega en dos pasos:

1. Agregar una asociación llamada To-Target en el diagrama de clases de navegación entre la clase del nodo Source y Target; aplicar la cardinalidad apropiada y la dirección de la asociación.
2. Agregar un ancla en la clase del nodo source para poder dar acceso al nuevo link.

La Figura 22 muestra un sketch de una porción del modelo de clases de navegación afectado por este refactoring, con la vieja versión a la izquierda y la nueva a la derecha.

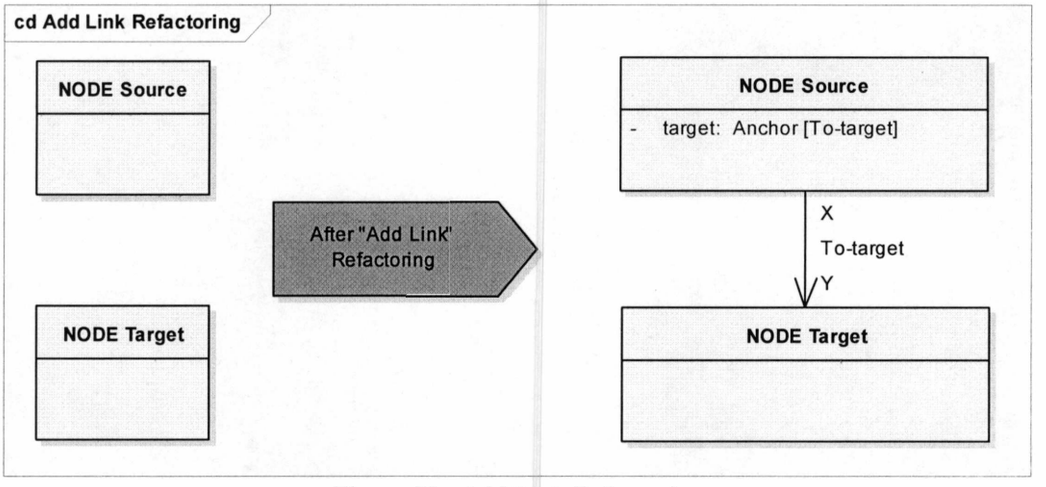


Figura 22 - Add Link Refactoring

3.10.3 Ejemplo

Hemos usado este refactoring en el ejemplo de "Split Node Class" para agregar links entre los nodos CD_Basic y CD_Details.

3.10.4 Refactorings relacionados

Un link puede ser agregar a un nodo transformando uno de sus atributos en un link; en este sentido este refactoring puede ser realizado por el "Turn Attribute Into Link".

Si el nuevo link es introducido para acortar el camino de navegación entre dos nodos existentes puede suceder que uno o más nodos intermedios queden en desuso, pudiéndose eliminar los mismos a través del refactoring "Remove Unreacheable Node".

Finalmente, si queremos mostrar este link con un widget en la interfaz gráfica debemos usar el refactoring de presentación "Add Interface Anchor".

3.11 Remove Redundant Link

3.11.1 Motivación

Permitir limpiar el modelo de navegación de links que salen del mismo origen y van al mismo destino sobre otros que se usan más a menudo.

3.11.2 Mecanismo

Sobre el nodo que queremos aplicar el refactoring buscamos el link que se desee eliminar, borramos el link y el ancla asociado de la definición del nodo.

La Figura 23 muestra un diagrama genérico de una porción del modelo de clases de navegación afectado por este refactoring, con la vieja versión a la izquierda y la nueva a la derecha.

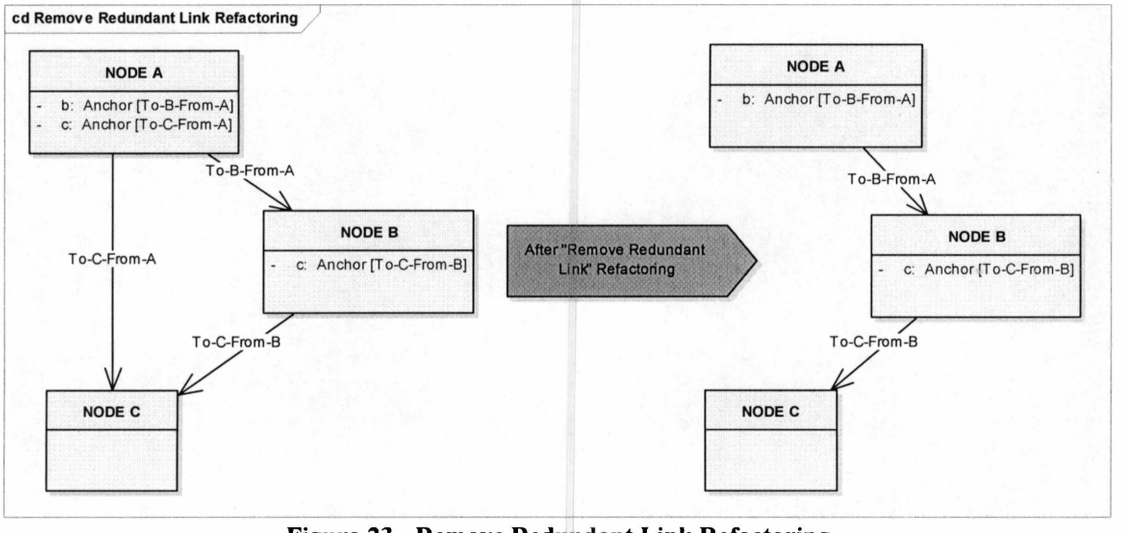


Figura 23 - Remove Redundant Link Refactoring

3.11.3 Ejemplo

En nuestro caso de estudio, podemos ver que desde el nodo Orden de compra hay dos caminos de navegación hacía el CD, uno a través del Item de la orden de compra o accediendo directamente desde el link hacía el CD

Podemos determinar que el usuario cuando revisa la orden de compra revisa los ítems y puede cambiar la cantidad de los mismos, y revisar el CD que acaba de comprar a través del link en Item. Podríamos eliminar el link que sale desde la orden de compra y va directamente a los CDs que contiene.

En la Figura 24 se ve el resultado del refactoring luego de aplicarse en nuestro caso de uso.

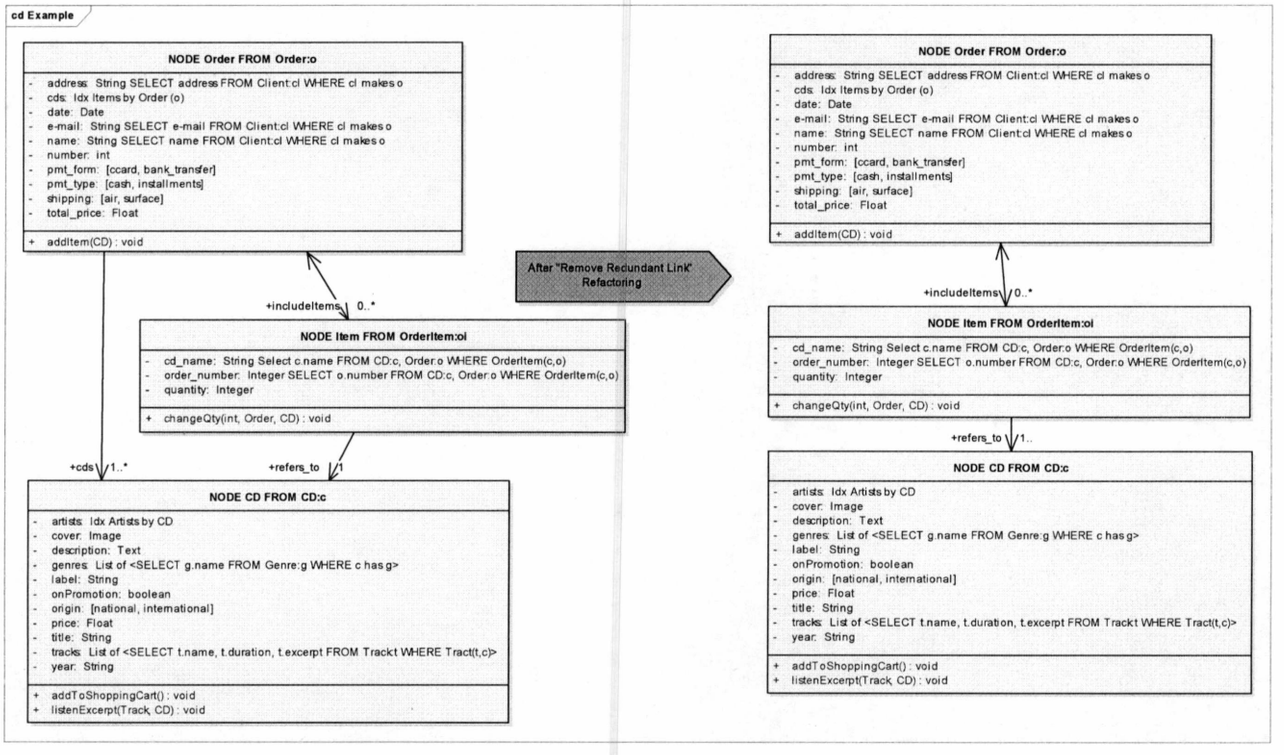


Figura 24 - Remove Redundant Link Refactoring - Caso de estudio

3.11.4 Refactorings Asociados

Luego de aplicar este refactoring se debería eliminar el widget que dispara el link desde la interfaz.

3.12 Add Index

3.12.1 Motivación

Es común que un sitio Web a medida que crece y se hace más popular, crezca en funcionalidad como en número de páginas. Debemos proveer alguna estructura de navegación para organizar el contenido de las páginas relacionadas para lograr que el usuario pueda encontrar de manera ordenada e intuitiva el contenido del sitio. Las estructuras más comunes que se utilizan para organizar la información son los índices.

Cuando el sitio crece puede ser necesario tener que agregar un índice.

Otra motivación para este refactoring es la aplicación del patrón Web "Browsable Content" [19] que en su motivación también preescribe organizar el contenido el categorías que tengan significado para los usuarios.

3.12.2 Mecanismo

La precondición de este refactoring es que todas las partes involucradas en la navegación ya deben estar presentes en el modelo de aplicación, la clase de elementos en el conjunto y el atributo que relaciona los elementos.

Un nuevo índice sobre un conjunto de elementos S organizados por un atributo A se agrega al diagrama navegacional en dos pasos:

Si no hay un índice existente que agrupe los elementos en S,

1. Agregamos un recuadro con línea entrecortada e indicamos el tipo de orden que vamos a definir sobre el atributo A, que puede ser "alfabético", "cronológico" o de acuerdo a la popularidad, etc.
2. Agregamos un recuadro con línea entrecortada e indicamos el nombre del atributo A el cual se están agrupando los elementos de S.
3. Aplicamos el refactoring "Add Link" entre tipo de orden y el nodo S.
4. Aplicamos el refactoring "Add Link" entre el Indice del atributo y el tipo de orden.

La Figura 25 muestra un sketch de una porción del diagrama de navegación afectado por este refactoring, con la vieja versión a la izquierda y la nueva a la derecha.

Figura 25 - After Add Index Refactoring

3.12.3 Ejemplo

En nuestro ejemplo de la tienda de compras de CDs, hay varios criterios para navegar sobre los CDs. Cuando se selecciona navegar los mismos, se puede elegir el genero "POP", pero por ejemplo, no se puede navegar de acuerdo al país de origen, se podría agregar un índice adicional para permitir navegar los CDs por país.

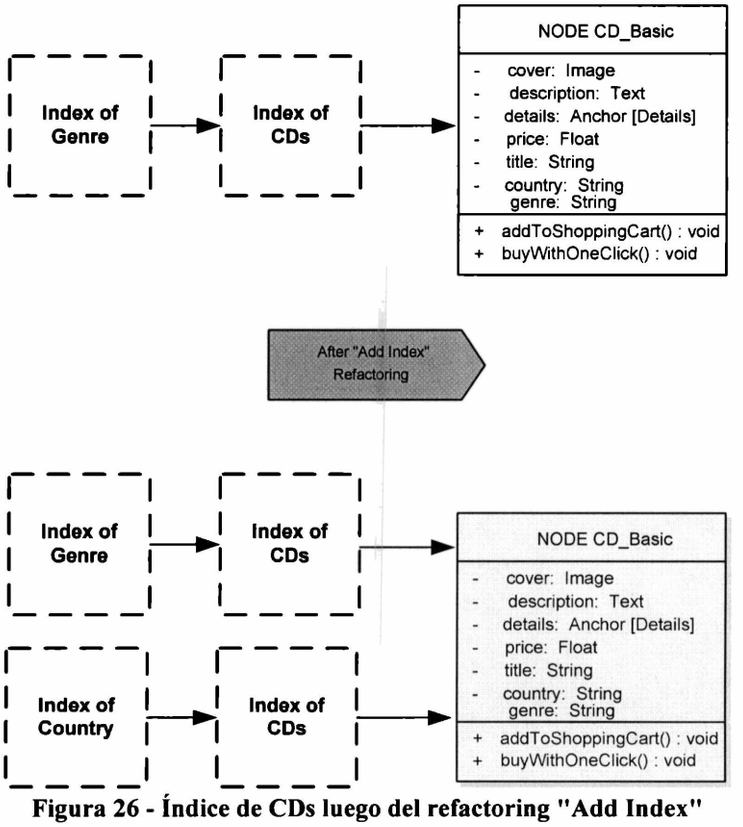


Figura 26 - Índice de CDs luego del refactoring "Add Index"

Dado que este es un refactoring de modelo de navegación, el impacto que tendría sobre la presentación debería ser agregar un widget para que el nuevo índice se muestre al usuario y permita navegar hacia los elementos.

A continuación se muestra un ejemplo de cómo quedaría el antes (Figura 27) y el después (Figura 28) de aplicar el refactoring "Add Index" luego de aplicar los cambios pertinentes en la interfaz.

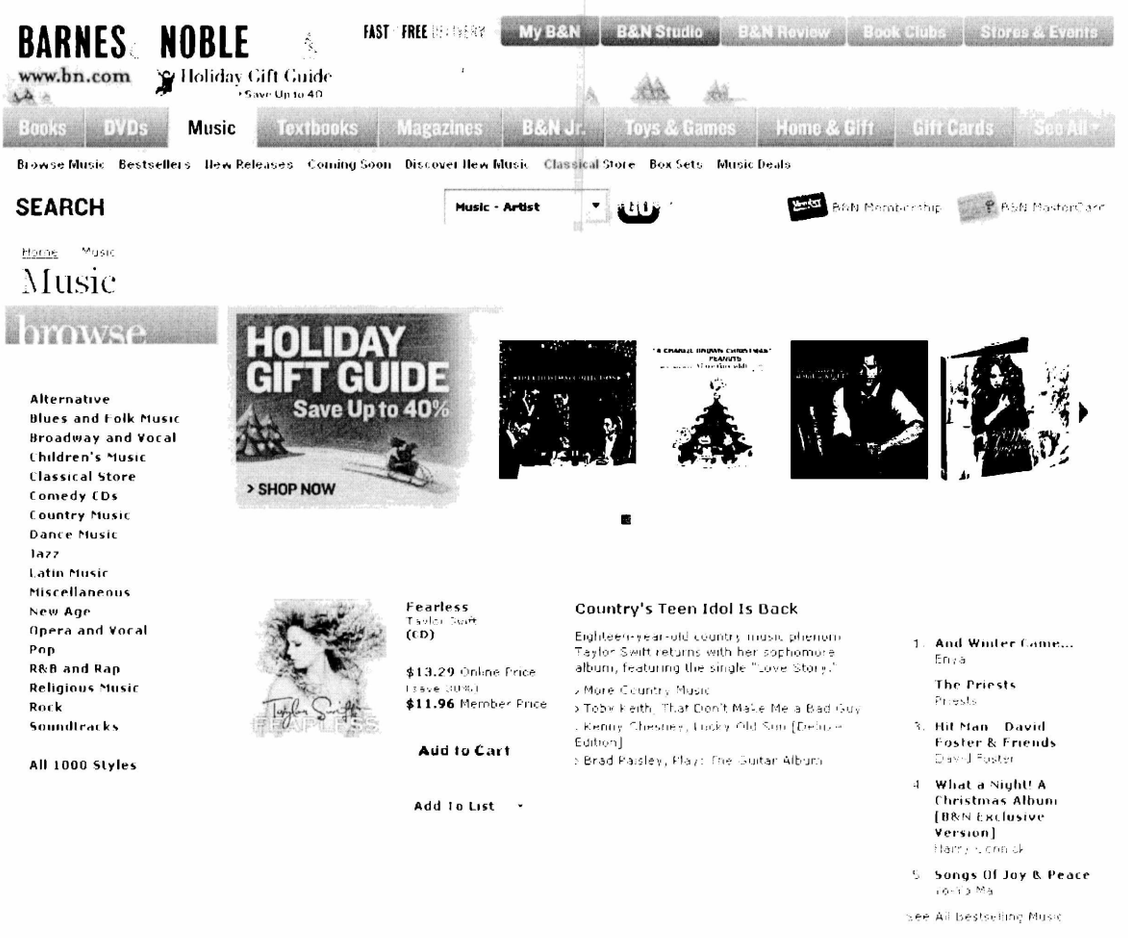


Figura 27 - Refactoring Add Index – Interfaz gráfica antes de aplicar el refactoring.

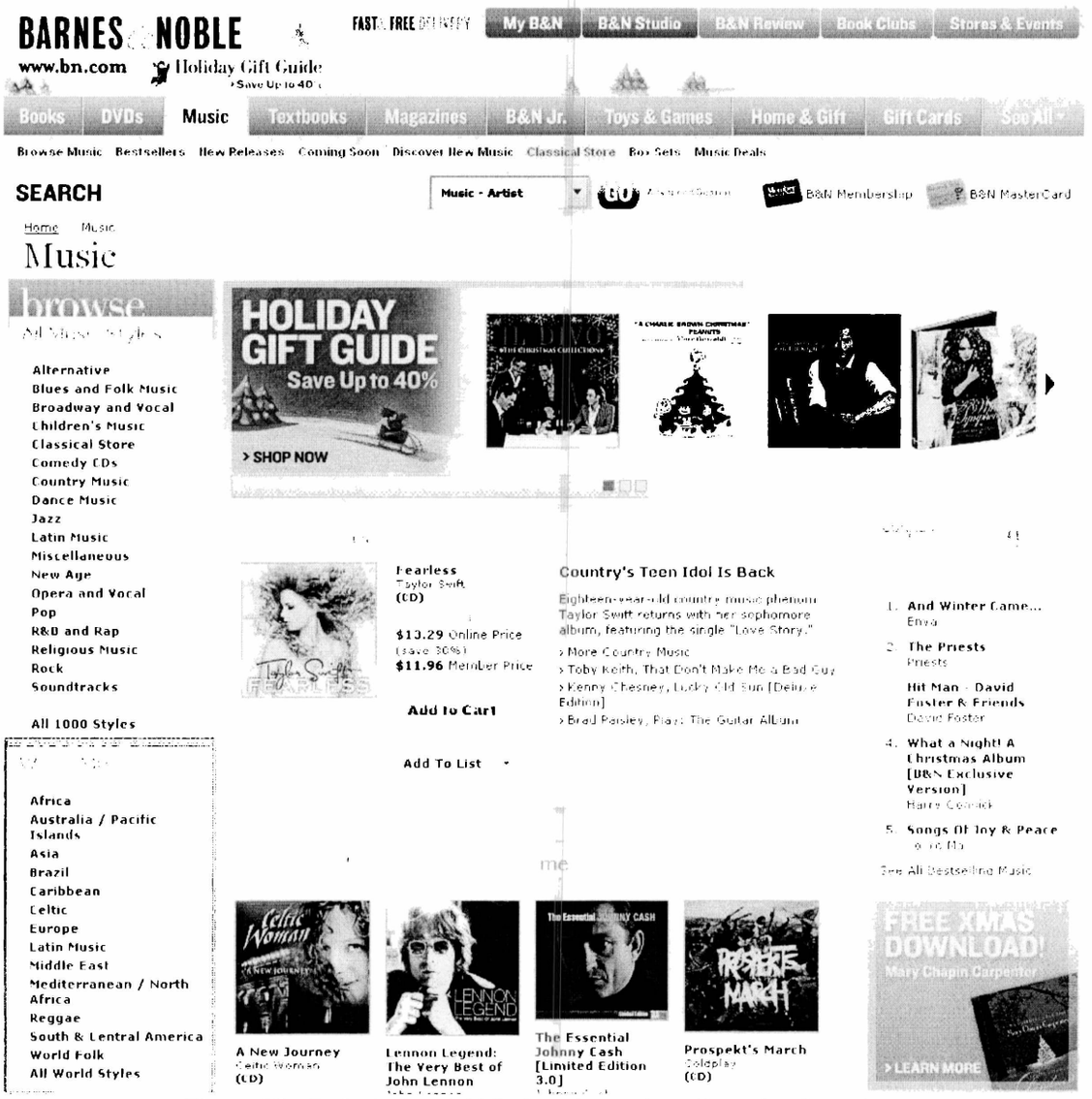


Figura 28 - Refactoring Add Index - luego de aplicar el refactoring.

3.12.4 Refactorings relacionados

Luego de aplicar este refactoring se puede aplicar el refactoring "Add Page Section" para mostrar el nuevo índice del diagrama de navegación.

3.13 Split Index

3.13.1 Motivación

Cuando un sitio Web incrementa su número de elementos, los índices existentes que agrupan demasiados elementos, pueden hacer que el usuario pierda tiempo buscando entre demasiadas entradas..

Otra motivación es la introducción del patrón "Hierarchical Organization", este patrón establece que los elementos dentro de un conjunto deben ser menor que cincuenta. Si el número de nodos en un índice crece por sobre este número, es conveniente dividir el índice de la categoría en un número de subcategorías.

3.13.2 Mecanismo

De manera similar al refactoring "Split Node Class", este es un refactoring compuesto, dado que dividir un índice involucra agregar nuevos subíndices a través del refactoring "Add Index". Aún más, índices a las subcategorías pueden ser agregados a través del refactoring "Add Index". De manera alternativa, si el refinamiento ocurre a nivel del modelo de la aplicación el índice asociado es definido como jerárquico.

1. Seleccionamos el índice a dividir.
2. Creamos un nuevo índice (agregamos un recuadro de línea entrecortada) indicando en el nombre el criterio por el cual estamos indizando.
3. Todos los links que llegan al índice original deben modificarse para que ahora lo hagan al nuevo índice agregado en dos.
4. Aplicamos el refactoring "Add link" entre el nuevo nodo y el nodo a ser dividido.

Cabe mencionar que luego de que se agrega el nuevo índice, los elementos a indizar por el nodo a dividir serán menores dado que el criterio de indización que se agrega con el nuevo índice disminuye la cantidad de elementos del mismo. A medida que se navega a través de los índices aplican condiciones de "y" entre los criterios.

La Figura 29 muestra un sketch de una porción del diagrama de navegación afectado por este refactoring, con la vieja versión a la izquierda y la nueva a la derecha.

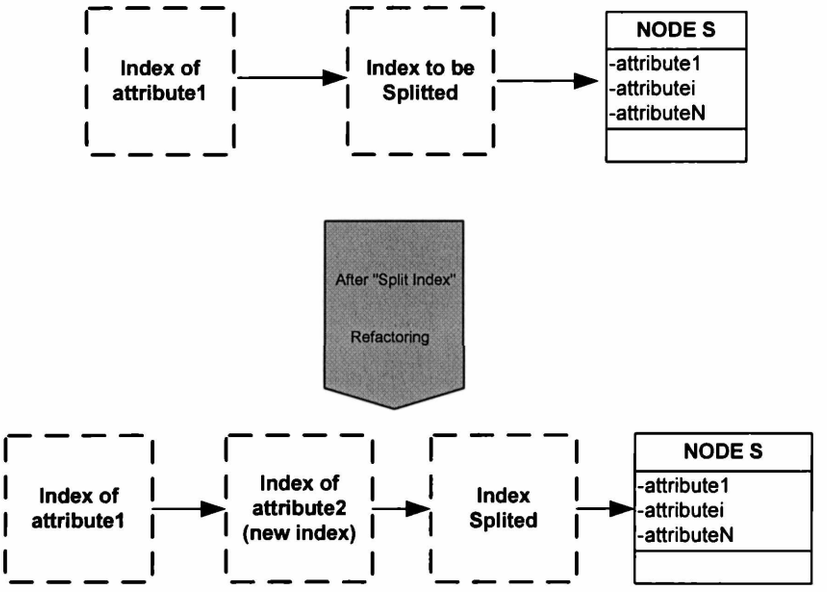


Figura 29 - Split Index Refactoring

3.13.3 Ejemplo

Los productos en un tienda virtual deben ser fáciles de encontrar, generalmente esto se hace a través de índices. Al mismo tiempo, una lista de productos en un índice no debe crecer mucho porque se convierte en difícil de leer y los consumidores se frustran fácilmente. Barnes & Nobles tiene una larga cadena de categorías que le permiten al usuario acceder a una gran variedad de libros.

Mientras se aplica este refactoring, el desarrollador debe tener en cuenta que agregar una nuevo nivel de subíndices, mientras se define la navegación, aumenta el camino de navegación sobre el contenido del sitio, por lo tanto, se debe encontrar el balance correcto entre la creación de subíndices y hacer el camino al producto lo más corto posible.

Continuando con nuestro ejemplo, hemos visto que el sitio de Barnes & Nobles nos permite navegar por género, en particular cuando elegimos pop, no existe una categoría dentro de "POP", que nos permita seleccionar el país de origen.

Así como en el ejemplo de "Add Index" agregamos la categoría por país, esta vez la agregaremos como subcategoría de Pop.

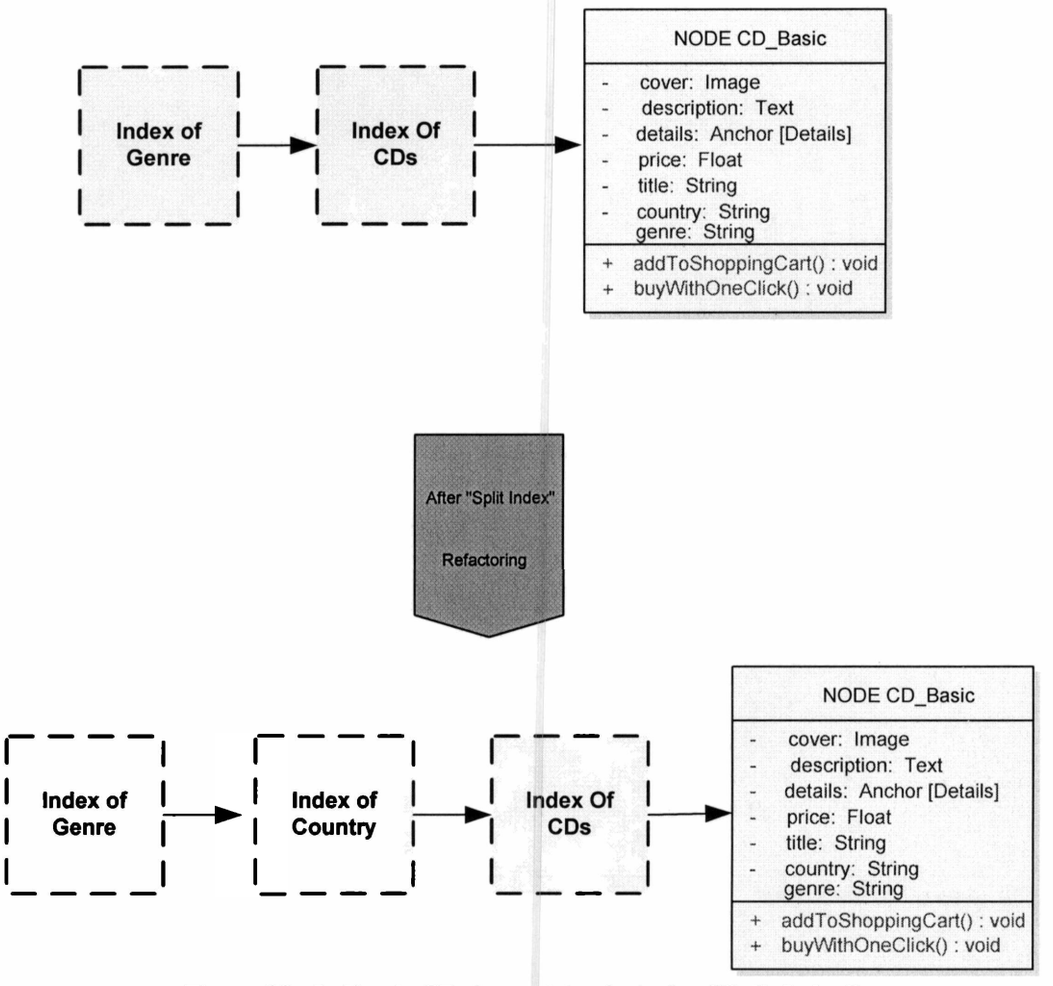


Figura 30 - Indice de CDs luego del refactoring "Split Index"

Luego de aplicar el refactoring en el modelo navegacional debemos realizar refactorings en el modelo de presentación para que el usuario pueda ver estos cambios. Dado que agregamos un nuevo índice, se debe agregar un widget para mostrar dicho índice dentro de la índice que acabamos de crear. Cabe destacar que aparte del widget debemos crear todas las páginas que resultaran de clickear en alguno de los elementos del índice.

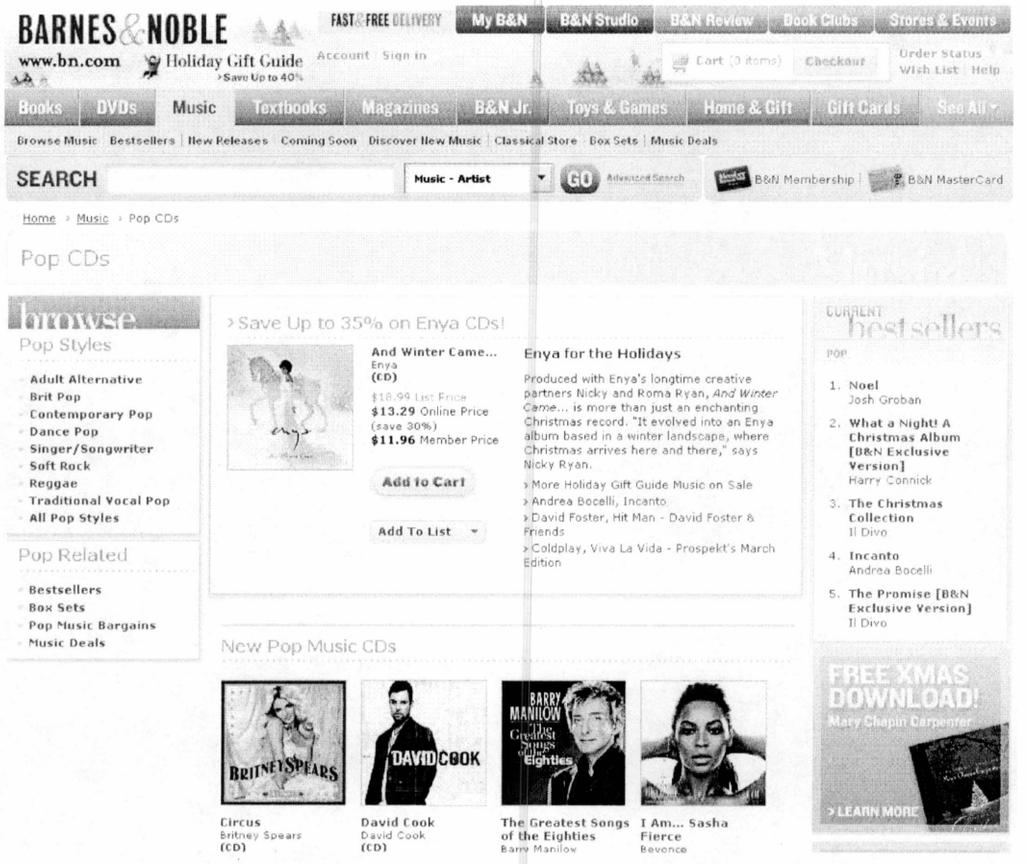


Figura 31 - Split Index refactoring – Categoria Pop antes del refactoring.

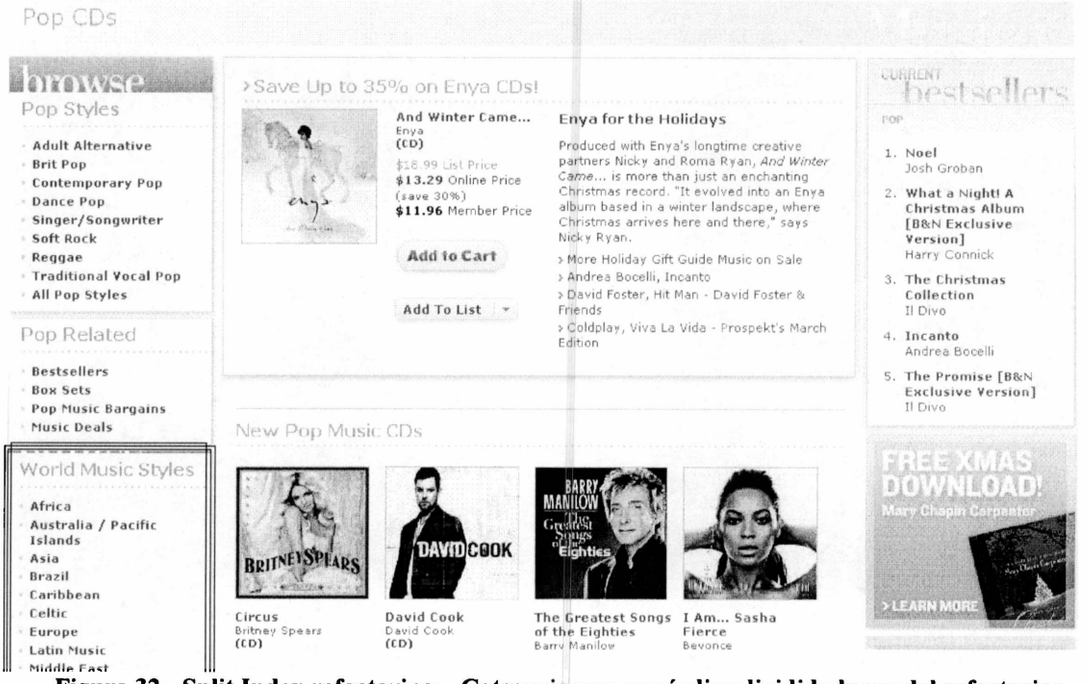


Figura 32 - Split Index refactoring – Categoria pop con índice dividido luego del refactoring

3.14 Merge Index

3.14.1 Motivación

La motivación es la inversa a la del refactoring anterior, luego de analizar el sitio nos damos cuenta que tener demasiadas categorías puede resultar confuso al usuario al momento de buscar, o porque existen categorías que redundantes y el patrón de comportamiento del usuario final en el sitio nos indica que ciertas categorías caen en el desuso.

Podemos unir categorías preexistentes.

3.14.2 Mecanismo

Como prerrequisito ambos índices debe existir en el modelo navegacional sobre un nodo S. Además se debe soportar esta navegación desde el modelo de la aplicación.

Primero debemos identificar el nodo a unir, generalmente será el resultado de la combinación de varios índices previos. Ver Figura 33.

A continuación eliminamos el primer índice previo al nodo a unir y movemos la condición de indización al índice anterior al índice eliminado.

Al mover la condición debemos elegir si:

1. Eliminamos la misma, en este caso, se "elimina" una categoría, pero el índice a unir queda con más elementos.
2. Unimos las condiciones con un "&" (y), lo que hace es simplemente acortar el camino de navegación.
3. Unimos las condiciones con un "||" (o), resultando en el incremento del número de elementos del índice que se ha unido.

La Figura 33 muestra un sketch de una porción del diagrama de navegación afectado por este refactoring.

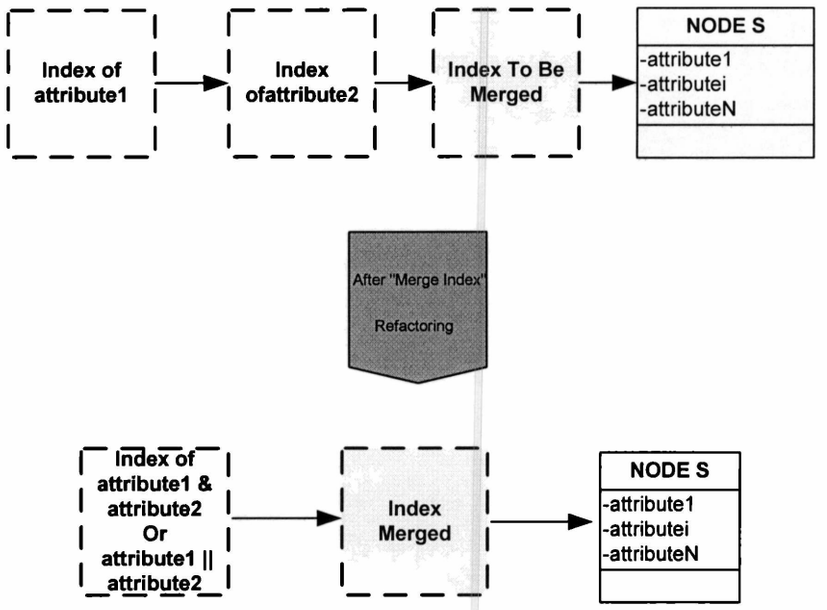


Figura 33 - Merge Index Refactoring

3.14.3 Ejemplo

Nuestro ejemplo puede ser el del refactoring anterior, al determinar, por ejemplo a través de un estudio en la ventas, que el sitio es usado básicamente por gente que vive en Estados Unidos y el 95% de las ventas son sobre CDs que se generan en ese país. Podríamos determinar que el índice por país es obsoleto y por lo tanto el índice alfabético que resulta de la navegación por género y país, se transforma en sólo por género, incrementando la cantidad de elementos en el índice alfabético resultante.

3.15 Relate Index

3.15.1 Motivación

Si dos íncies tienen elementos en común, los consumidores deberían ser capaces de navegar de una a otra y viceversa. Usando este refactoring proveemos "MultipleWays to Navigate", para prevenir que el sitio se vuelva tedioso de usar.

3.15.2 Mecanismo

La precondición de este refactoring es que el modelo de navegación ya posea definiciones de los índices de navegación a ser relacionados. Luego, el refactoring es realizado usando el refactoring "Add Link" para crear un link bidireccional entre los indices.

El procedimiento se realiza según lo muestra la Figura 34, primero debemos determinar que índice se relaciona con cual, y luego agregar un link entre ambos índices.

La Figura 34 muestra un sketch de una porción del diagrama de navegación afectado por este refactoring, con la vieja versión la parte superior y la nueva en la inferior.

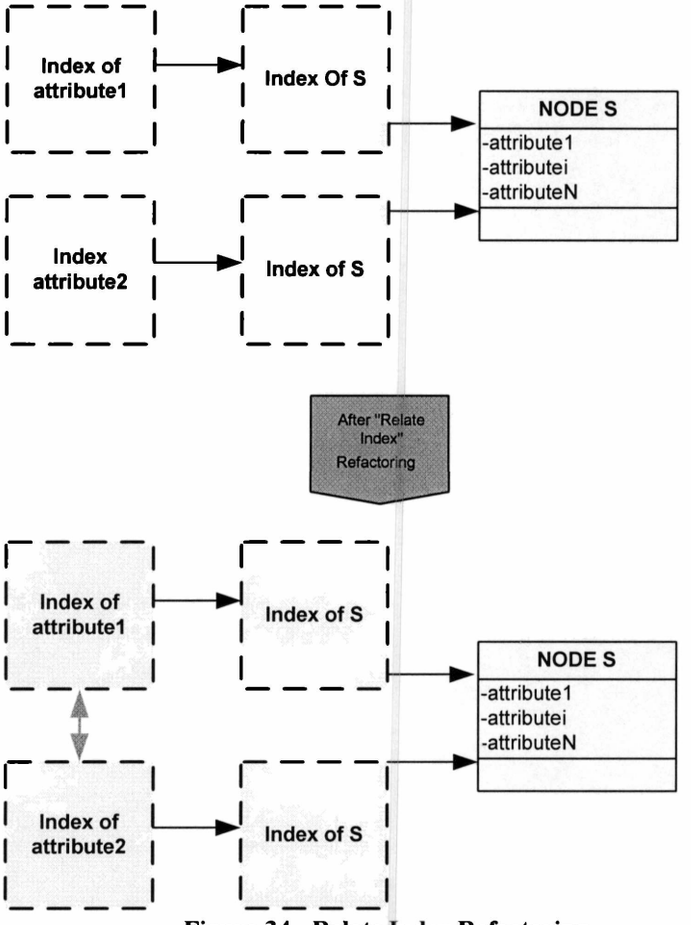


Figura 34 - Relate Index Refactoring

3.15.3 Ejemplo

En muchos sitios populares como Barnes & Nobles, Amazon y otros, la clasificación de los ítems de venta están dado por lo que en principio parecen ser categorías, pero encontramos que muchos ítems se encuentran a través de navegar de muchas maneras diferentes y diferentes categorías. Cuando decimos categorías nos referimos a formar de agrupar la información a través de índices.

Por ejemplo, si en el sitio Barnes & Nobles buscamos el libro "cien años de soledad" ver Figura 35, podemos ver que está clasificado en dos categorías distintas, "Awards" y en al menos seis subcategorías distintas dentro de "Fiction"; esto nos lleva a pensar que la organización dentro de estos sitios hace que en vez de tener un artículo categorizado en una sola categoría haya muchas, por ende se definen "alias" de la misma categoría, a través de tags, y luego se relacionan de acuerdo a la forma en que el usuario está más familiarizado.

De esta manera dado que las categorías se encuentran relacionadas, debemos proveer al usuario maneras de navegar a través de las mismas.

Figura 35 - Mapa de categorias para "100 años de soledad"

3.15.4 Refactorings relacionados

Luego de aplicar el refactoring en el modelo navegacional debemos realizar refactorings en el modelo de presentación para que el usuario pueda ver estos cambios. El refactoring asociado es "Add Anchor Interface" que permité agregar un link entre los dos índices.

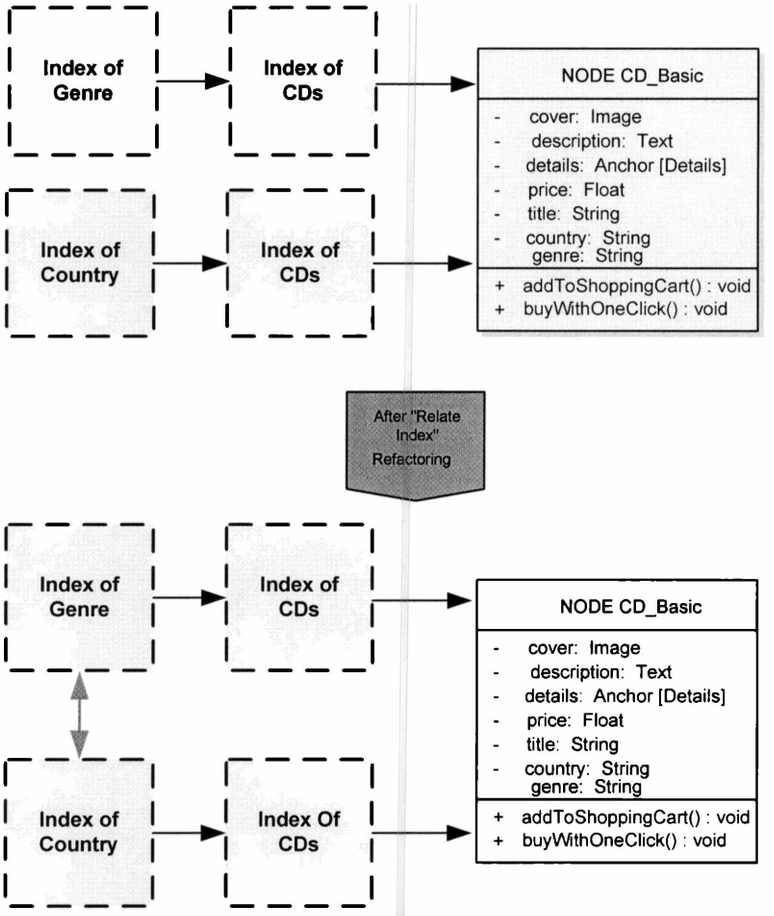


Figura 36 - Relate Index Refactoring - Ejemplo

Capítulo 4
Refactorings de Modelo de Presentación

Según la metodología OOHDM en los diagramas de presentación se definen diversos elementos que nos permiten modelar la presentación de una aplicación. Recordemos del capítulo dos que estos elementos se agregan en Abstract Data Views.

Los elementos que serán afectados por nuestros refactoring son:

- El tipo de los objetos de interfaz (widgets) que componen la página (ADV)
- El arreglo y composición de los wigdets en un ADV.
- Las transformaciones de interfaz que se suceden luego de la interacción del usuario.

Utilizaremos la definición dada por [24] para definir un refactoring de modelo de presentación.

Definición:

Un refactoring de presentación puede cambiar los elementos definidos en el modelo de presentación pero debe preservar:

- El conjunto de operaciones disponibles y su semántica como está definida en el modelo de aplicación y navegación.
- La disponibilidad de un elemento de la interfaz, es decir que el refactoring no removerá del ADV un widget que deje una operación, ancla o link, o todo un nodo del modelo de navegación sin su vista en el modelo de presentación. Aún así puede ser divido o unido y los widgets pueden ser reemplazados.

Los refactorings del modelo de presentación pueden:

- Cambiar el tipo de los widgets por otro, pero conservando la funcionalidad subyacente.
- Cambiar la disposición de los widgets en los ADV.
- Dividir o unir ADVs.
- Agregar información u operaciones a una página, siempre y cuando el modelo de navegación y de aplicación lo soporten.
- Cambiar los efectos de la interfaz

A continuación se describen los refactorings del modelo de presentación. La metodología elegida consistirá en explicar la motivación, su mecanismo con un ejemplo genérico y luego con un ejemplo de una aplicación del mundo real.

4.1 Casos de estudio.

Para mostrar los refactorings de presentación utilizaremos los siguientes casos de estudio:

- Facebook: la red social más popular y con mayor cantidad de usuarios en el mundo.
- El sitio de e-commerce de Tematika, el sitio de venta de libros de librerías populares en Argentina como "El Ateneo", Yenny y eXtra.
- El sitio de e-commerce de Cuspide.com.ar, una editorial argentina de libros.
- El sitio de e-commerce de Musimundo.
- El correo electrónico de Google GMail, actualmente una de las aplicaciones Web más utilizadas en el mundo con millones de usuarios
- El sitio de e-commerce de libros Barnes & Nobles, una de las mayores tienda de compra de libros on-line en el mundo.
- El sitio de e-commerce Amazon.com, una de las mayores tienda de compras on-line en el mundo.
- El sitio de e-banking del banco Santander Rio.

4.2 Split Page

4.2.1 Motivación

La motivación de este refactoring puede ser: como consecuencia de aplicar el refactoring de modelo de navegación "Split Node" o debido a que un nodo se ha vuelto demasiado atestado de información y mostrar toda la información del nodo en la misma página hace que el usuario se pierdan debido a que debe realizar demasiado scrolling.

Lo que propone este refactoring es partir o dividir la página en una o más páginas o secciones.

4.2.2 Mecanismo

Supongamos que queremos dividir una página en dos. Los pasos a seguir son los siguientes:

1. Agregar una página nueva que tenga un ADV vacío para la misma.
2. Usar el refactoring "Move Widget" para mover los widgets seleccionados (widgets de información y de acción) desde la página original a la nueva.
3. Usar el refactoring "Add Interface Anchor" en cada página para agregar links entre las dos páginas y permitir al usuario acceder al contenido y las operaciones disponibles en la página original.
4. Verificar si es necesario cambiar el nombre del ADV original y si es así cambiarlo.

La Figura 37 muestra un sketch de una porción del diagrama del modelo de presentación afectado por este refactoring, con la vieja versión arriba y la nueva abajo.

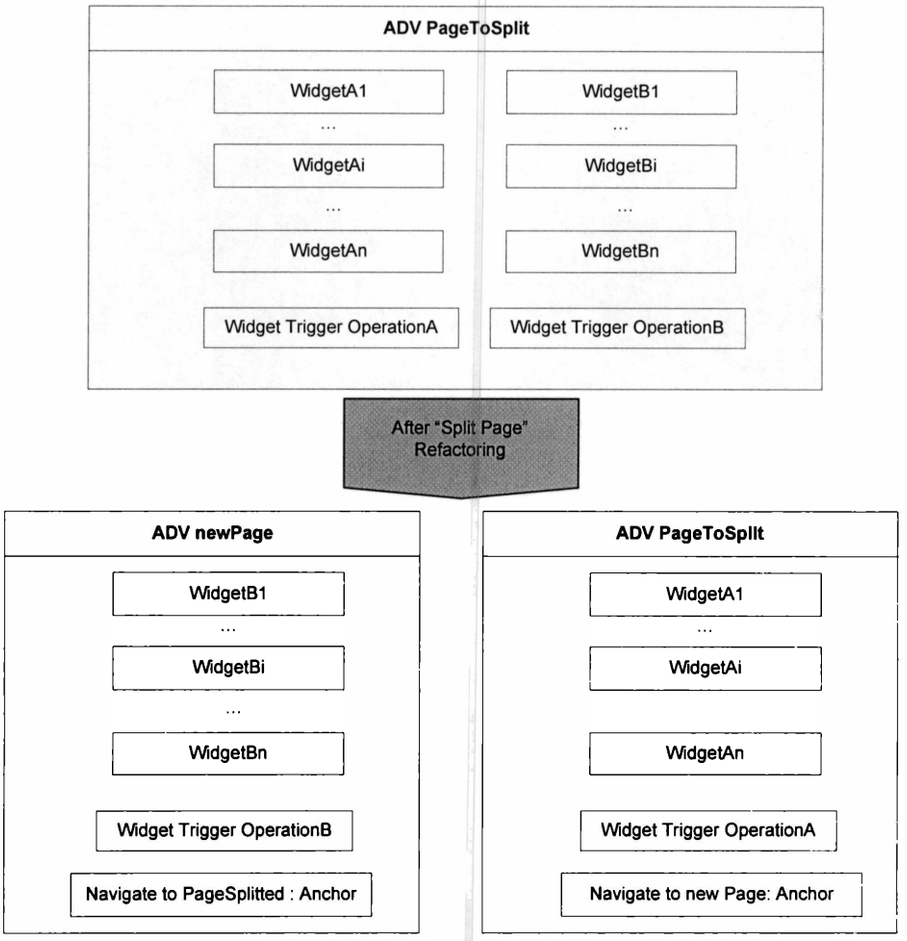


Figura 37 - Split Page Refactoring

4.2.3 Ejemplo

Facebook fue pasando por muchos cambios hasta ser la aplicación que es hoy. En la Figura 38 se puede ver la antigua página principal del Facebook de un usuario luego de ingresar al sitio.

La página de la Figura 38 que muestra el perfil del usuario, posee una sección llamada "Mini-Feed" que muestra la información sobre las acciones que realizó el usuario sobre items publicados por otros usuarios, como por ejemplo, comentar una foto, cambiar el estado de lo que está haciendo actualmente, etc. También posee una sección donde se listan los amigos del usuario, más la información personal del usuario, entre otras.

En la Figura 39 se muestra como quedó la página principal del usuario de Facebook, luego de las modificaciones introducidas por la empresa. Se puede ver que la

sección del Mini-Feed fue movida a una página nueva que es accesible a través de la solapa "Wall" que se puede apreciar en la figura; también la información personal, como las fotos del usuario fueron organizadas en páginas diferentes accesibles mediante los tabs, "Info" y "Photos".

En este ejemplo se ve claramente como la página del perfil del usuario se saturó de información y fue necesario crear nuevas páginas que se hicieron accesibles mediante tabs. También se pueden ver ejemplos del refactoring "Add Section Page"

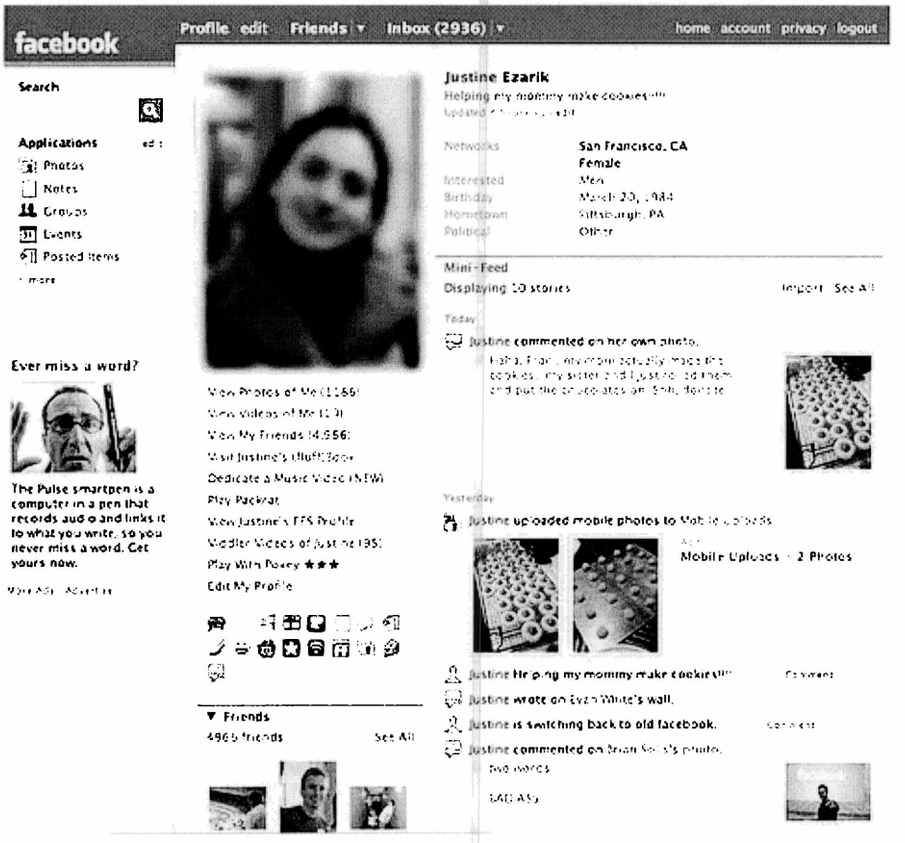


Figura 38 - Antiguo Facebook - Pagina principal de un usuario

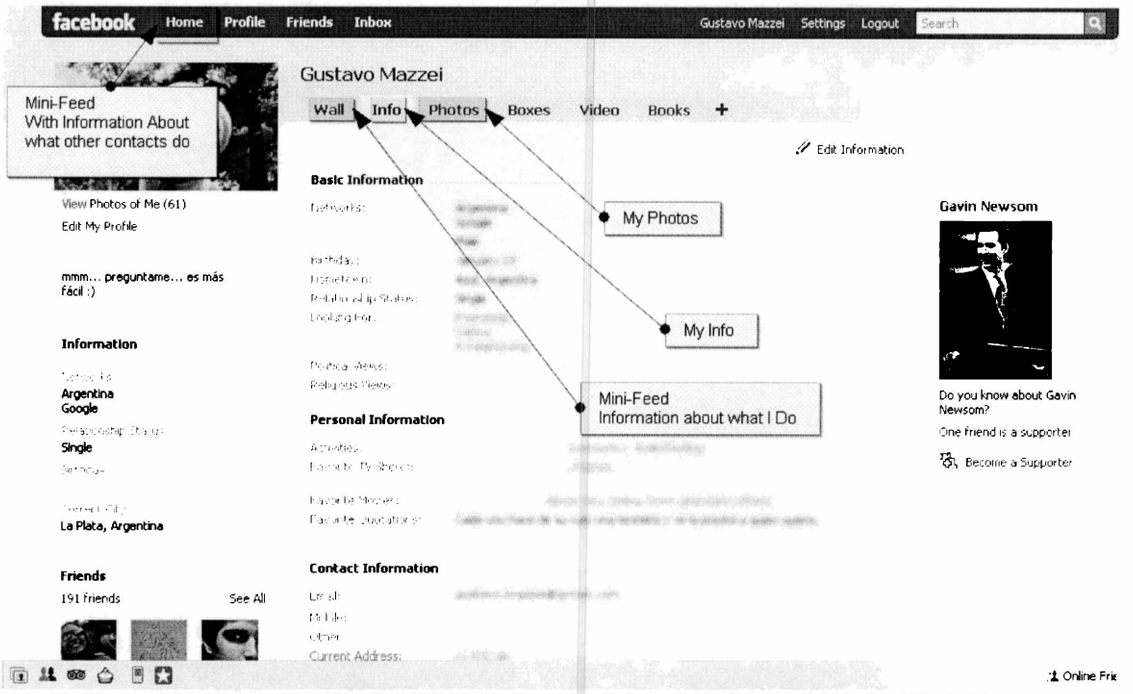


Figura 39 - Nuevo Facebook - Página principal del usuario luego de varios "Split Page"

4.3 Merge Pages

4.3.1 Motivación

Este refactoring está motivado principalmente por la aplicación del refactoring de navegación "Merge Node", o cuando se necesiten unir dos o más páginas en una sola.

4.3.2 Mecanismo

Supongamos que queremos unir dos páginas A y B. Los pasos a seguir son los siguientes:

1. Elegir la página a eliminar (supongamos B) y la página en la que se va a reunir el contenido de la otra (supongamos A).
2. Usar el refactoring "Move Widget" para mover los widgets de información y de acción desde la página B a la página A.
3. Eliminar el ADV de la página B.
4. Verificar si es necesario cambiar el nombre del ADV de la página A y si es así cambiarlo.

La Figura 40 muestra un sketch de una porción del diagrama del modelo de presentación afectado por este refactoring, con la vieja versión arriba y la nueva abajo.

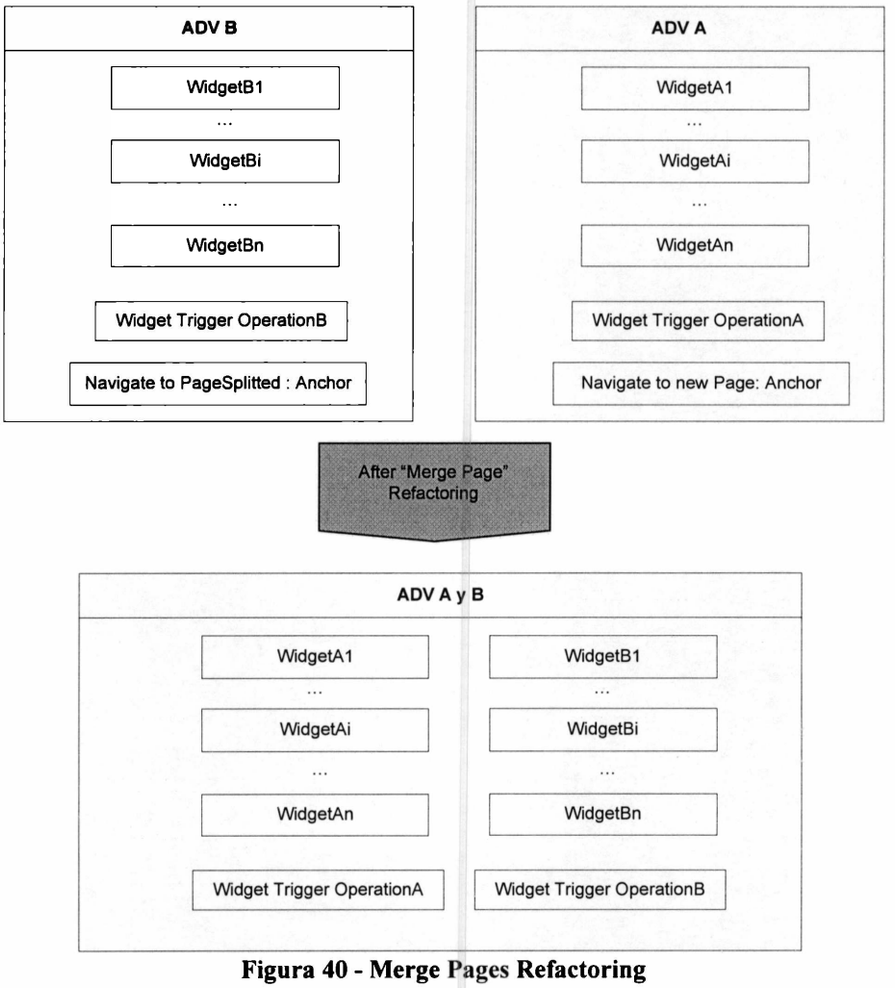


Figura 40 - Merge Pages Refactoring

4.3.3 Ejemplo

El proceso de check-out del sitio www.amazon.com se realiza en varios pasos, uno de ellos es la selección de la dirección a la cual se enviará la orden resultado de los productos que el usuario compra.

En las Figura 41 y Figura 42 se muestra dos de los pasos de check-out del sitio y en la Figura 43 se muestra el resultado de aplicar este refactoring. Como se ve el usuario ahora puede seleccionar el tipo de envío en la misma página que elige la dirección a la cual se enviará la orden.

Este refactoring es posible puesto que la página uno (Figura 41) no está muy saturada de información. Después de este refactoring puede aplicarse el "Add Page Section" si la información que se está agregando pertenece a lo que podríamos denominar una sección nueva.

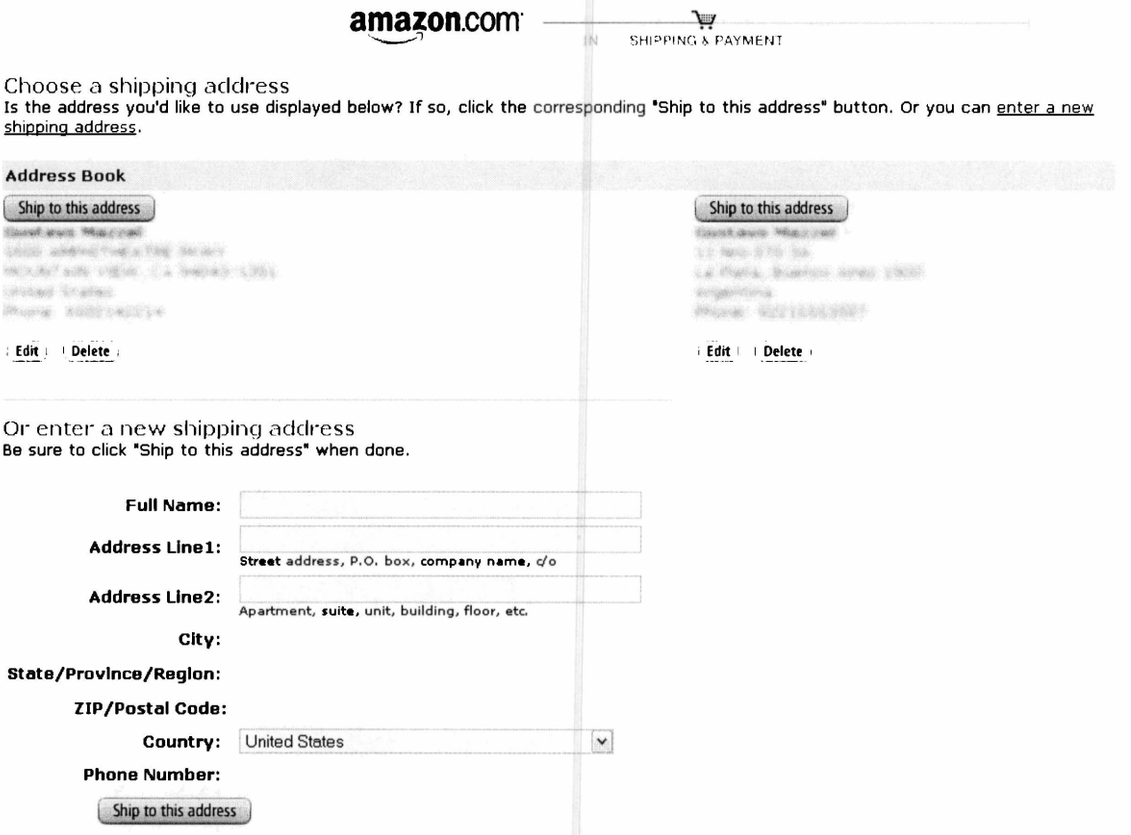


Figura 41 - Página de proceso de checkout de Amazon.com (1)

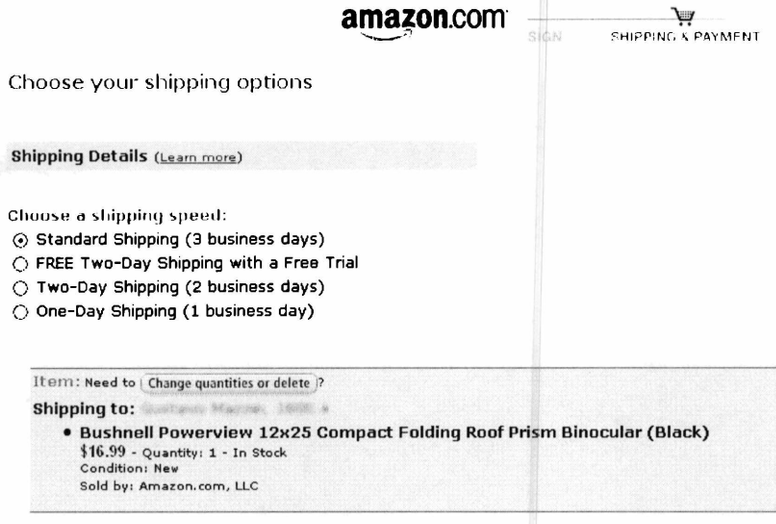


Figura 42 - Página de proceso de checkout de Amazon.com (2)

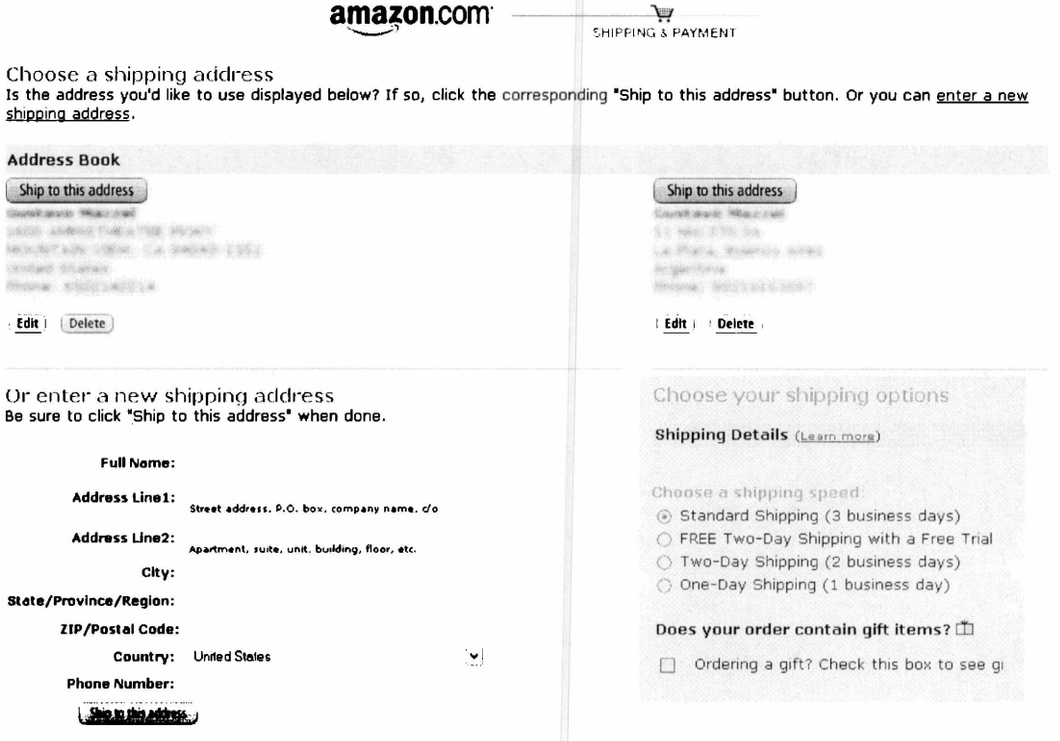


Figura 43 - Página de proceso de checkout de Amazon.com - Luego del refactoring merge pages

4.4 Add Page Section

4.4.1 Motivación

La motivación de este refactoring es permitir agregar una sección (porción de una página que puede ser representada por un ADV interno en el modelo de presentación de OOHDM) a una página existente, por ejemplo, para mostrar información de un nodo adicional introducido a través "Add Node Class" en el modelo de navegación.

4.4.2 Mecanismo

Este refactoring puede realizarse en los siguientes cuatro pasos.

1. Definir ADVs internos para cada sección.
2. Usar el refactoring "Move Widget" para mover los widgets de la interfaz del ADV en la página al ADV en la sección correspondiente.
3. Usar el refactoring "Add Interface Anchor" en el ADV que describe la página, tantas veces como nuevas secciones se hayan definido, para permitir la navegación desde la parte superior de la página a cada sección.
4. Usar el refactoring "Add Interface Anchor" al final de cada sección para permitir navegar hacía la parte superior de la página.

La Figura 44 muestra un sketch de una porción del diagrama del modelo de presentación afectado por este refactoring, con la vieja versión a la izquierda y la nueva a la derecha.

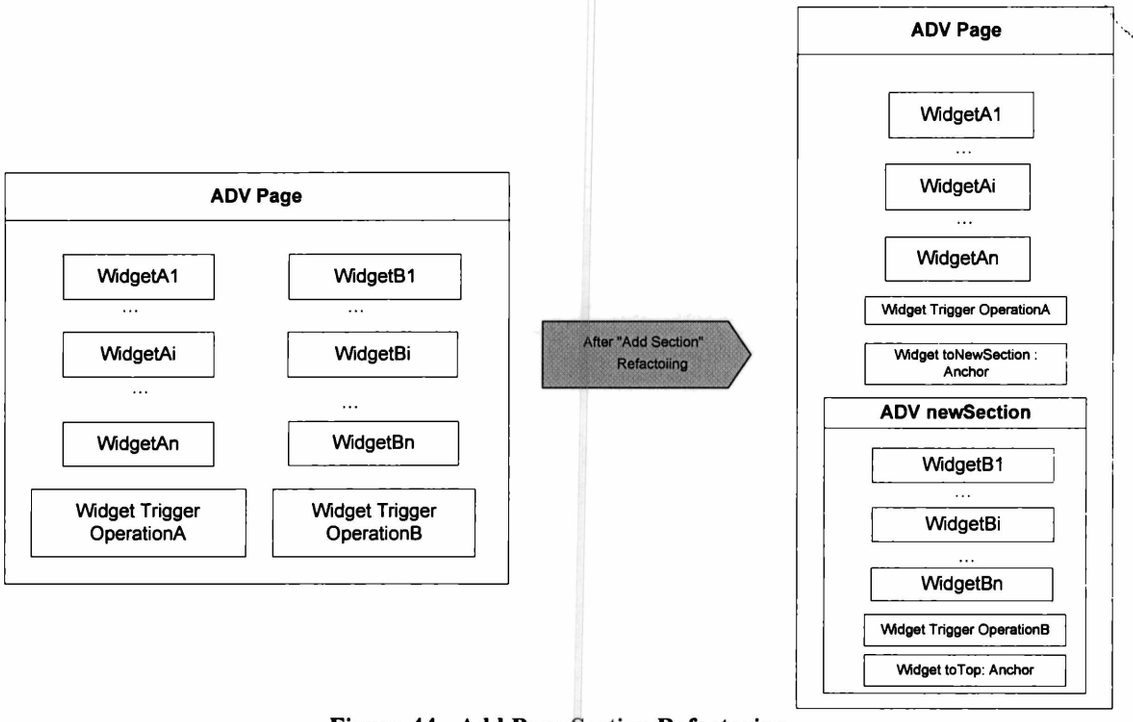


Figura 44 - Add Page Section Refactoring

4.4.3 Ejemplo

En la Figura 45 se puede ver el sitio antes del refactoring. A la derecha de la imagen se puede ver la sección "Information" que representa la información del usuario tal como nombre, información básica de contacto, etc.

En la Figura 46 se puede ver la misma sección en el nuevo Facebook, donde fue divida en diferentes subsecciones, ya que se ampliaron las opciones de privacidad por sección.

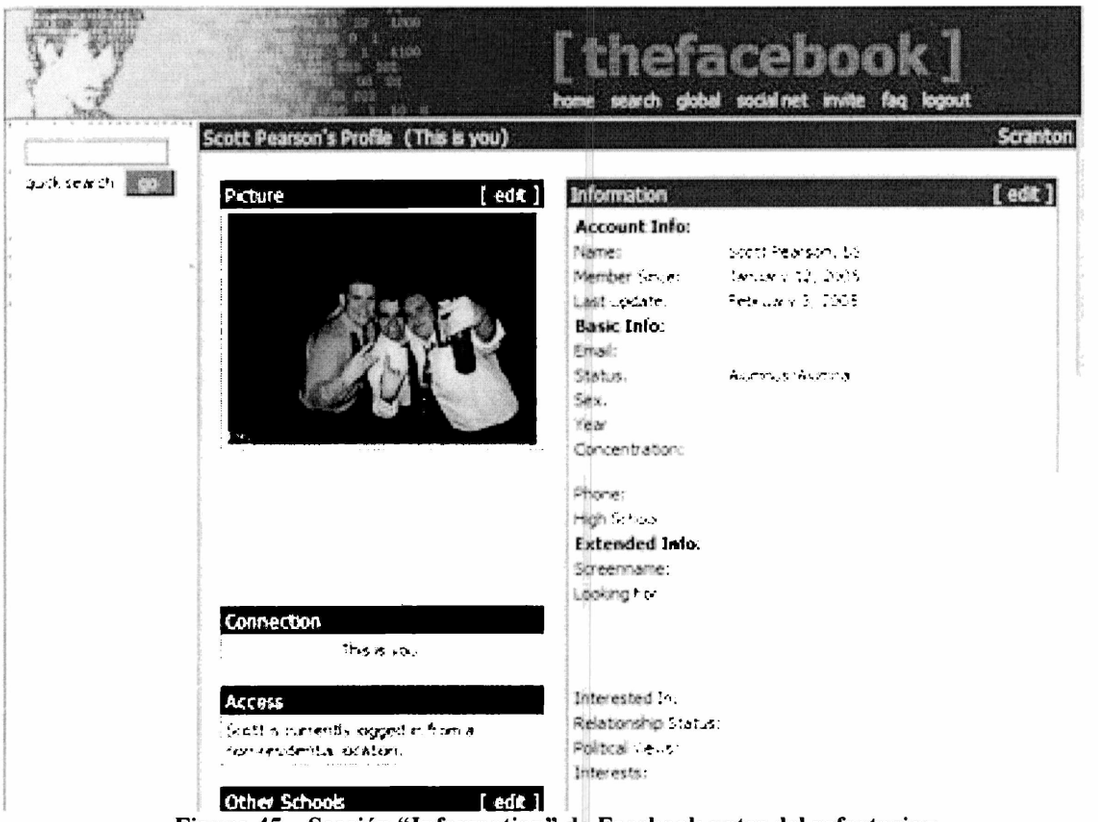


Figura 45 – Sección "Information" de Facebook antes del refactoring

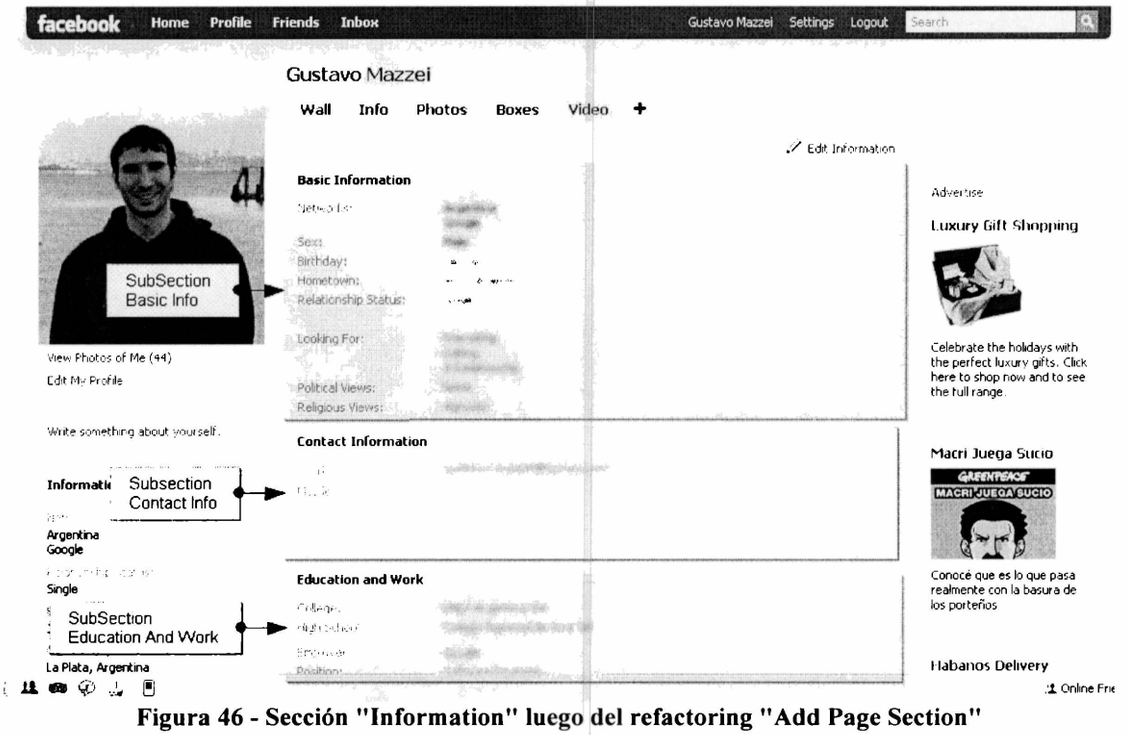


Figura 46 - Sección "Information" luego del refactoring "Add Page Section"

4.5 Move Widget

4.5.1 Motivación

Este refactoring permite mover un widget de presentación/interacción de una página a otra, por ejemplo, como resultado de un "Split Page", algunos widgets necesitan ser movidos de la página original a la nueva.

4.5.2 Mecanismo

Este refactoring se realiza en los siguientes tres pasos:

1. Se selecciona el widget de ADV de la página de origen.
2. Se copia la definición del mismo al ADV de destino, puede ser un ADV interno o de una página nueva.
3. Se elimina el widget del ADV de origen.

La Figura 47 muestra un sketch de una porción del diagrama del modelo de presentación afectado por este refactoring en el caso del ADV interno.

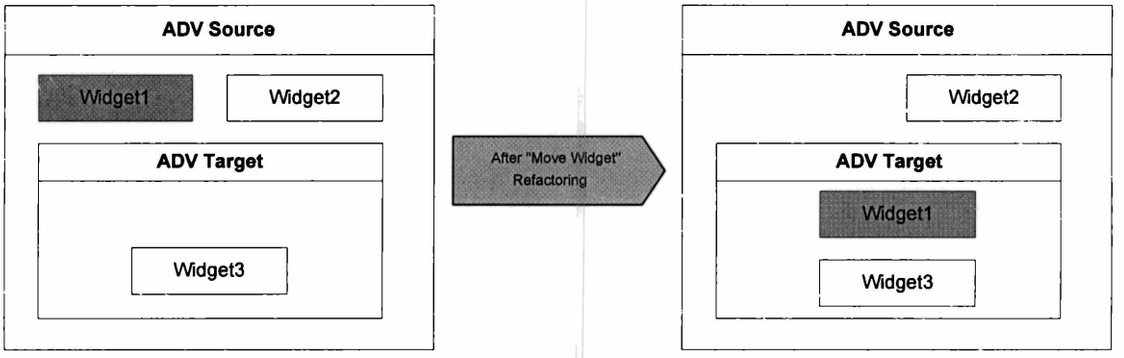


Figura 47 - Move Widget Refactoring – Ejemplo Interno ADV

La Figura 48 muestra el caso del refactoring en caso de que el ADV de destino sea un ADV que representa una página diferente

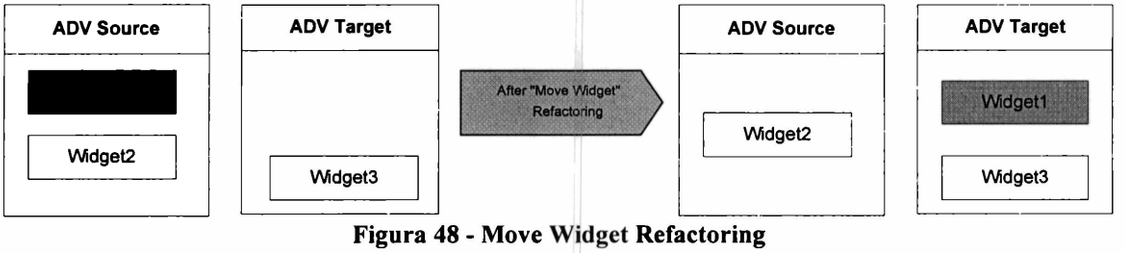


Figura 48 - Move Widget Refactoring

4.5.3 Ejemplo

En la Figura 38 y la Figura 39 se muestra el antes y el después del cambio de diseño de la página de Facebook, ahí se puede ver claramente la aplicación del refactoring, algunos ejemplos son: el mini-feed, la información sobre los contactos, fueron movidos desde el ADV original al ADV que representa la pestañas de Wall y Friends respectivamente.

4.6 Replace Widget

4.6.1 Motivación

Un análisis del uso del sitio puede mostrar que el tipo de widget usado para mostrar algún ítem de información o para activar una operación debe ser cambiado por otro más apropiado, para mejorar la operabilidad, usabilidad o accesibilidad.

4.6.2 Mecanismo

La precondición para este refactoring en el caso del widget que activa una operación es que el nuevo tipo de widget no cambie la funcionalidad subyacente de la operación. Por ejemplo, una lista de selección simple no puede ser cambiada por una de selección múltiple (al menos que cambie el modelo de la aplicación). Si la precondición se mantiene, se debe identificar el ADV para la página del modelo de presentación que contiene el widget en cuestión y reemplazarlo por el más apropiado.

4.6.3 Ejemplo

Los check-box están pensados para permitir al usuario realizar operaciones, seleccionando uno o más elementos de una lista. Un típico ejemplo de su uso es en una aplicación de correo electrónico, donde se permite seleccionar correos individuales y luego aplicarles operaciones como borrarlos, marcarlos como leídos, etc. Por lo tanto, el usuario no espera que al momento de marcar un check-box se dispare una operación. Este

es el caso del sitio www.cuspide.com en el cual luego de agregar varios ítems al carro de compras y ver su contenido (ver Figura 49), podemos ver la lista de ítem con un check-box a la izquierda y al seleccionar un elemento de la lista (ver Figura 50), el comportamiento esperado es que se marque el ítem y no que se dispare la operación de borrado sobre este elemento. Esto puede ser confuso para el usuario en términos de comportamiento esperado, por lo tanto podemos aplicar nuestro refactoring reemplazando los check-box por una imagen que sea más intuitiva para el usuario, en este caso un cesto de basura. La Figura 51 muestra como quedaría la página luego de este refactoring.

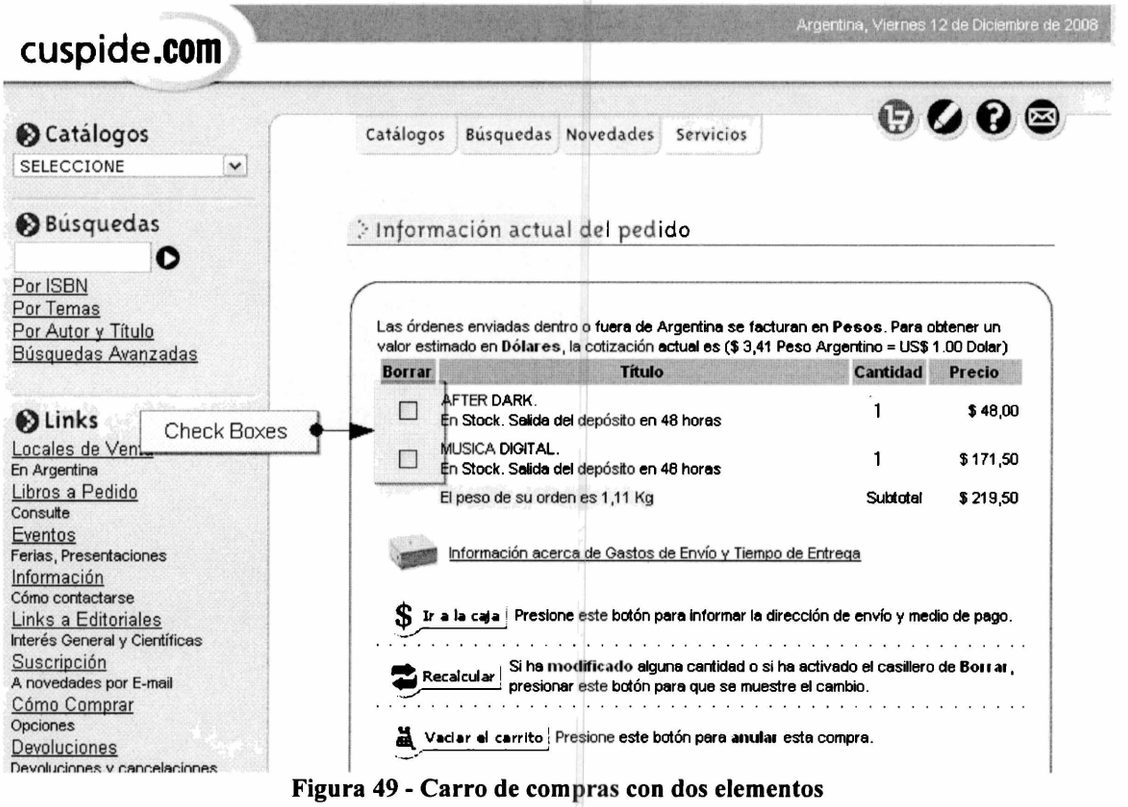


Figura 49 - Carro de compras con dos elementos

Figura 50 - Carro de compras luego de checkear el último checkbox de la lista.

Figura 51 - Carrito de compras luego del refactoring "Replace Widget"

4.7 Add Interface Anchor

4.7.1 Motivación

Este refactoring es consecuencia del refactoring de navegación "Add link" o "Turn Attribute Into Link", o bien, cuando se necesita hacer visible un link de un nodo de navegación.

4.7.2 Mecanismo

El mecanismo para realizar este refactoring consta de dos pasos:

1. Identificar el ADV asociado al nodo de navegación que ha sido afectado por el refactoring "Add Link" o bien, el que posea el ancla a activar.
2. Agregar el widget en el ADV que dispare active el ancla del nodo.

4.7.3 Ejemplo

GMail. actualmente una de las aplicaciones web más utilizadas en el mundo, con millones de usuarios. A pesar de que no ha cambiado significativamente su funcionalidad principal, ha realizado varias integraciones con otras aplicaciones de Google, como por ejemplo, con los documentos, fotos, etc. En la Figura 52 se puede ver una porción de cómo era la antigua página de GMail antes de la integración con el resto de las aplicaciones.

En la Figura 53 podemos ver una porción de la nueva página de GMail, con sus nuevos links agregados en la parte superior de la aplicación que nos permiten navegar hacia el resto de las otras aplicaciones.

Este es un claro ejemplo del refactoring "Add Interface Anchor".

Figura 52 - GMail antes de la integración con el resto de las aplicaciones de google

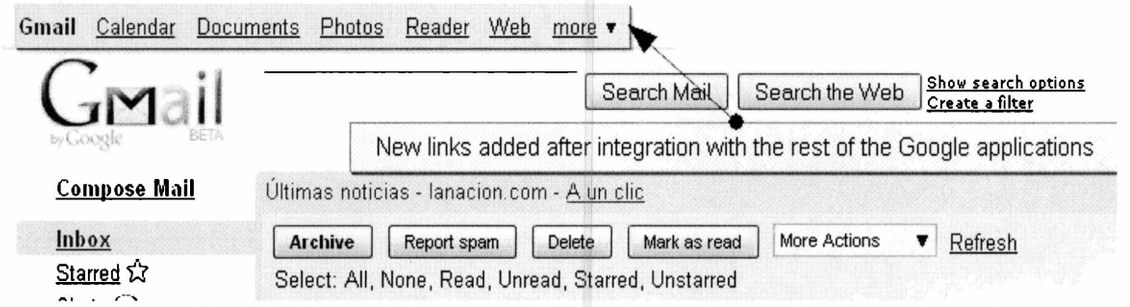


Figura 53 - GMail luego de la integración con el resto de las aplicaciones de google

4.8 Add Processing Page

4.8.1 Motivación

La motivación de este refactoring es la de resolver el problema de mantener al usuario esperando sin información sobre lo que está sucediendo luego de que se efectúa una operación.

Generalmente cuando el usuario dispara una operación que puede llevar varios minutos de procesarse, por ejemplo en los sitios de Home Banking, reservas de vuelos y transacciones electrónicas, si el sitio no provee una manera de indicar al usuario que su operación se está realizando, este puede disparar esta operación varias veces al no notar ningún cambio en la pantalla. Esto puede traer consecuencias importantes ya que un usuario disparando varias veces la misma operación reiteradas veces, puede tener efectos negativos sobre la operación que se intenta realizar

Básicamente este refactoring consiste en agregar información acerca del progreso de la operación, usualmente es una barra de proceso, o un reloj de arena mostrando el progreso de la operación accionada por el usuario

4.8.2 Mecanismo

El mecanismo de este refactoring depende del tipo de aplicación:

Si es una aplicación RIA utilizaremos una agregación de ADVs, haremos que un ADV de procesamiento se haga visible luego de que el usuario haga click sobre el botón o link que dispara la operación, y cuando la misma termine el ADV se ocultará y se mostrará el ADV de los resultados.

La Figura 54 muestra el template del ADV que se usará para introducir la página de procesamiento. Este ProcessingPageADV se instanciará con dos elementos, el ADV del formulario de ingreso de datos y el botón que disparará la operación. Cuando la operación es completada, se navega a la página de resultados.

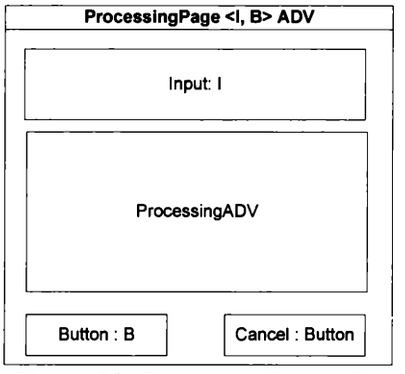
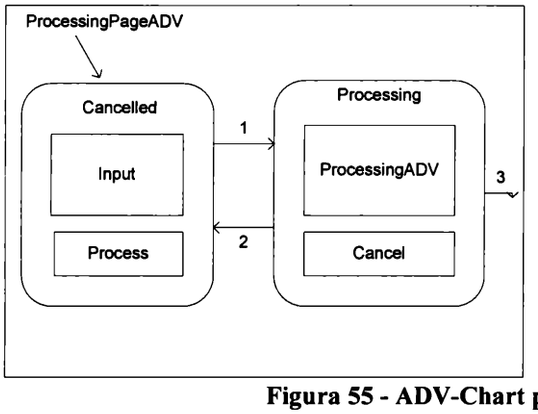


Figura 54 - Processing Page ADV



1:
Event: MouseClicked
Pre-Cond: Focus(Process)
Post-Cond: owner.isOperationInProgress() &&
perCont=perCont-Input+ProcessingADV

2:
Event: MouseClicked
Pre-Cond: Focus(Cancel)
Post-Cond: owner.isOperationCancelled() &&
perCount=perCont-ProcessingADV+Input

3:
Event: operationDone
Pre-Cond: owner.isResultReady()
Post-Cond: owner.navigateToResults()

Figura 55 - ADV-Chart para Processing Page ADV

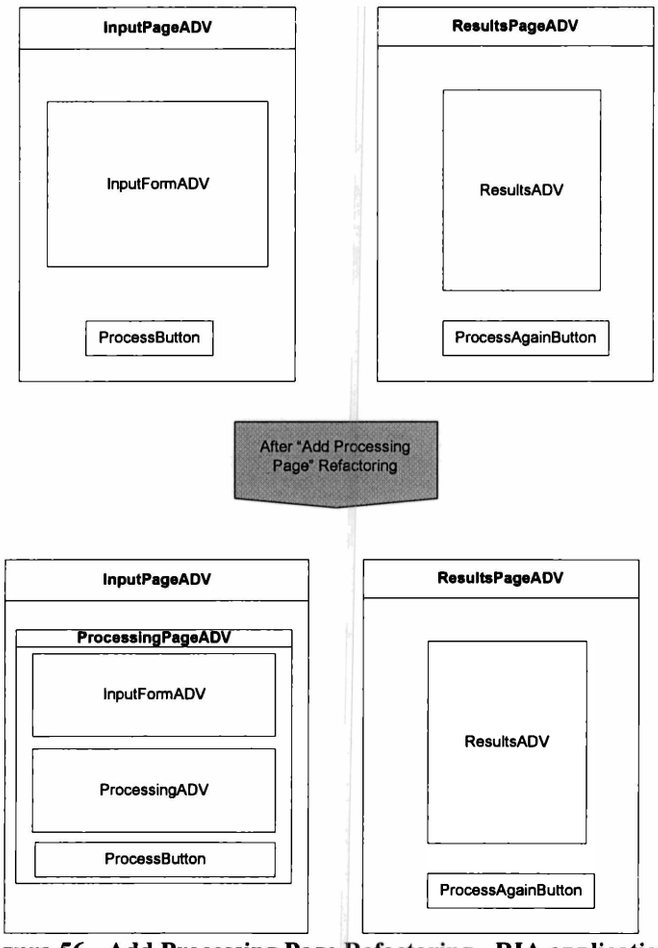


Figura 56 - Add Processing Page Refactoring - RIA application.

Si es una aplicación Web tradicional este refactoring es realizado en tres pasos básicos:

1. Crear una nueva página agregando un ADV vacío.
2. Agregar widgets en el nuevo ADV para la barra de progreso y probablemente un texto.
3. Cambiar la interfaz asociada con el widget que desencadena la operación, y que también navegue hacia la página de procesamiento. OOHDM usa ADV-charts, una variante de state-charts, para especificar los aspectos dinámicos de ADVs.

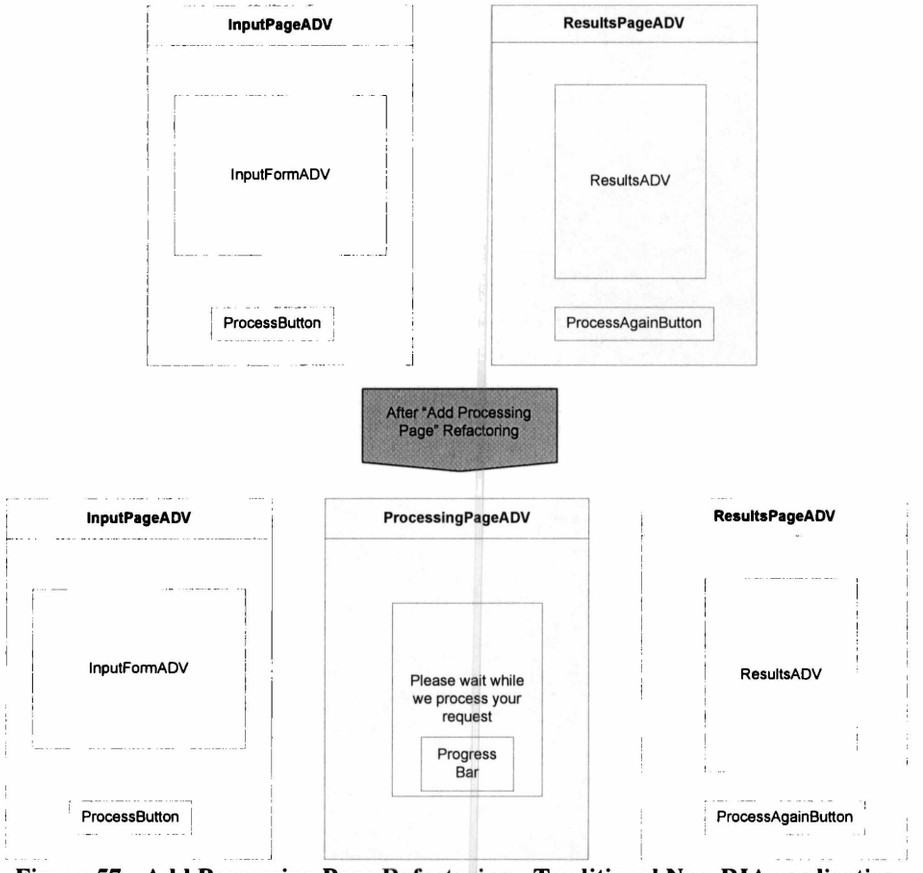


Figura 57 - Add Processing Page Refactoring - Traditional Non-RIA application

4.8.3 Ejemplo

Este tipo de refactoring es muy útil en sitios de banca electrónica, donde algunas operaciones pueden tardar cierta cantidad de tiempo y el usuario no tiene noción del progreso de la operación. Incluso tener un página que indique que la operación se está realizando evita que los usuarios clickeen varias veces sobre el link, evitando que la operación se dispare varias veces.

En el siguiente ejemplo usaremos al sitio www.santanderrio.com.ar, en la Figura 58 vemos la página principal del sitio luego de loguearnos, vemos que hay una operación sobre la tarjeta de crédito que es "ver último resumen", esta operación puede tardar varios minutos antes de mostrarnos resultados. La página de resultados puede verse en la Figura 60.

El resultado de aplicar el refactoring sería agregar la página que se muestra en la Figura 59.

Figura 58 - Página inicial de un sitio de banca electrónica.

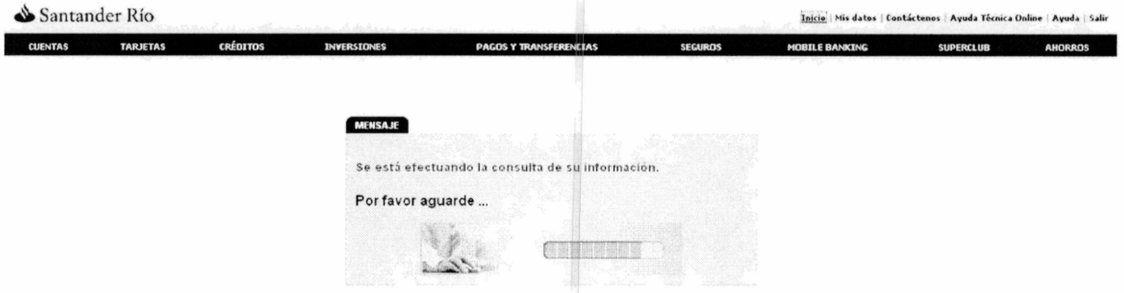


Figura 59 - Pagina de espera - procesamiento.

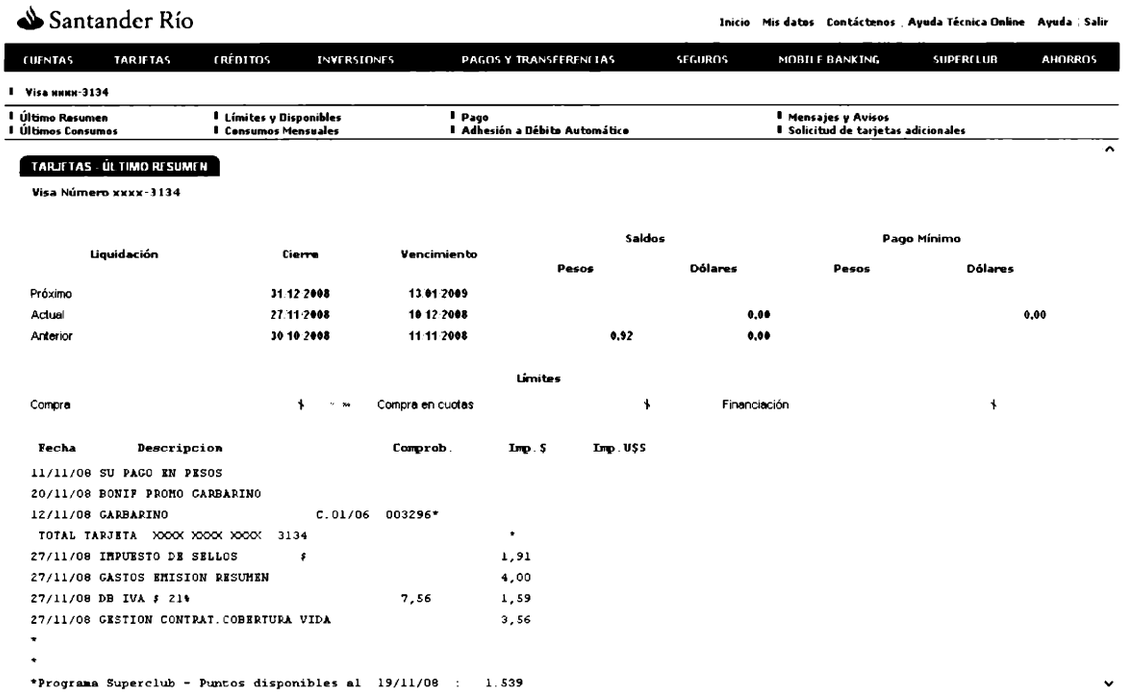


Figura 60 - Página de "Ultimo resumen" del sitio de banca electrónica.

4.9 Introduce Location

4.9.1 Motivación

Usualmente el los sitios de comercio electrónico los usuarios navegan buscando productos en diferentes categorías y subcategorías, a su vez un producto en una categoría puede llevar a buscar en otra categoría productos relacionados. Al moverse durante el sitio, el usuario puede perder el sentido de en qué categoría o subcategoría está.

Este refactoring ayuda al usuario a conocer constantemente la ubicación del producto dentro de la jerarquía de categorías y subcategorías que el sitio le adjudica.

4.9.2 Mecanismo

Como prerequisito para este refactoring el modelo de navegación debe tener en el nodo asociado al ADV una operación que nos devuelva la ubicación del nodo con respecto a la página de principal o "home".

Luego debe ubicarse el ADV responsable de mostrar los datos del producto o la lista de productos dentro de una categoría.

Usar el refactoring "Add Interface Anchor" por cada categoría o subcategoría, hasta llegar a la actual.

4.9.3 Ejemplo

A continuación podemos ver un ejemplo en el sitio www.musimundo.com conocido para hacer compras por Internet.

El sitio posee un menú a la izquierda donde se pueden seleccionar la categoría de lo que se puede comprar, y luego de hacerlo nos aparece a la derecha el contenido de la categoría, pero luego de hacer click en la categoría, el usuario no tiene información sobre la categoría que está mirando, ni como hacer para navegar a la categoría padre de la actual.

En el ejemplo que se muestra en la Figura 61, se buscó dentro del producto Música -> Infantiles.

Luego de este refactoring el usuario puede navegar a la categoría padre música haciendo click en el link de "Música" en el camino de navegación agregado, como se muestra en la parte inferior de la Figura 61.

TESINA DE LICENCIATURA

TITULO: Refactorings de los Modelos de Navegación y Presentación en Aplicaciones Web

AUTORES: Mazzei, Gustavo

DIRECTOR: Dr. Rossi, Gustavo

CODIRECTOR: Dra. Garrido, Alejandra

CARRERA: Licenciatura en Informática Plan 90

Resumen

Las aplicaciones Web evolucionan constantemente moldeándose al usuario, los procesos de desarrollo por excelencia son los procesos ágiles.

Estas metodologías iteran constantemente sobre las fases de desarrollo de la aplicación permitiendo que se introduzcan cambios en cada iteración. En este proceso la aplicación puede sufrir muchos tipos de modificaciones, algunas simples y otras más radicales.

Esta necesidad de cambio constante ha llevado a los desarrolladores a utilizar la técnica de refactoring, incrementando la calidad del diseño y del código mientras se preserva el comportamiento.

Indistintamente del proceso de desarrollo elegido, la mayoría de las metodologías de diseño para aplicaciones Web coinciden en tres modelos de diseño: de aplicación, de navegación y de presentación.

La técnica del refactoring, que en si fue concebida para reestructurar código, ahora se utiliza también a nivel de modelos.

En este trabajo me concentré en buscar las modificaciones más comunes que se realizan a aplicaciones Web y documentarlas de manera tal de formar un catálogo con los refactorings de modelo de navegación y presentación.

Con el creciente uso de los lenguajes orientados a objetos en el mercado y sobre todo en la tecnología Web, me pareció adecuado utilizar OOHDM para documentar los cambios en estos modelos.

Líneas de Investigación

Refactoring en aplicaciones Web.

Trabajos Realizados

Se provee un catalogo bien documentado de refactorings para el modelo de navegación y presentación de aplicaciones Web, mediante ejemplos resueltos sobre diagramas de navegación y presentación de OOHDM, mostrando su uso en aplicaciones del mundo real.

Conclusiones

Es posible mejorar de manera incremental y simple una aplicación Web mediante pequeños cambios y mejoras en sus modelos de navegación y presentación, sin cambiar los requerimientos básicos de la aplicación.

Se contribuye con un catalogo detallado de refactorings para el modelo de navegación y presentación, aplicados sobre aplicaciones del mundo real.

Trabajos Futuros

Desarrollo de un catalogo de refactorings de modelo de navegación que incluya los modelos diagramas de contexto de OOHDM.

Mostrar los refactoring en implementaciones Web conocidas como struts, GWT, etc.

Desarrollar herramientas para automatizar los refactoring de los modelos de presentación y navegación.

Desarrollar un catalogo para refactorings de modelos de presentación para aplicaciones RIA.

Fecha de la presentación: Abril, 2009

After "Introduce Location" Refactoring

Figura 61 - Introduce Location Refactoring

4.10 Provide Breadcrumbs

4.10.1 Motivación

Normalmente las aplicaciones Web permiten a usuario navegar, por las usualmente numerosas páginas que la conforman, lo cual puede llevar al usuario a "perderse" después de navegar varias de las mismas. Este refactoring ayuda al usuario a que esto no ocurra agregando a cada página un registro del camino de navegación que llevó al usuario hasta la página actual y permite volver a cualquier punto anterior en su recorrido, o bien, mostrar el camino desde la pagina principal hasta la ubicación del elemento actual.

Este refactoring consiste también en agregar en los típicos procesos por etapas, indicadores de la etapa actual del proceso.

4.10.2 Mecanismo

Este refactoring consiste en agregar un widget que nos indique en qué lugar del sitio estamos con respecto a una ubicación determinada y agregar un ancla de link al ADV que vaya en algunos casos a la página "Home", y en otros casos con respecto al inicio de un proceso de etapas o la página previa.

Como prerrequisito, el nodo asociado al ADV que muestra la información debe poseer una operación que le permita recuperar el camino de navegación hasta ese momento. Esta información puede ser obtenida guardando la historia de navegación del usuario.

Si es un proceso de etapas de varias páginas, los widgets que se agregan después del refactoring se activarán en secuencia e incrementalmente a medida que se avance en el proceso. Es decir, si navego de la página uno, luego a la dos y luego a la tres, en la página uno tendremos los tres widget hacia la página uno, dos y tres, pero no estarán activados para navegar hacia las mismas; cuando se navegue hacia la página 2, los widgets disponibles en esta página son el que navega hacía la página uno y el widget que activa el link hacia la página 3 estará desactivado; y así sucesivamente.

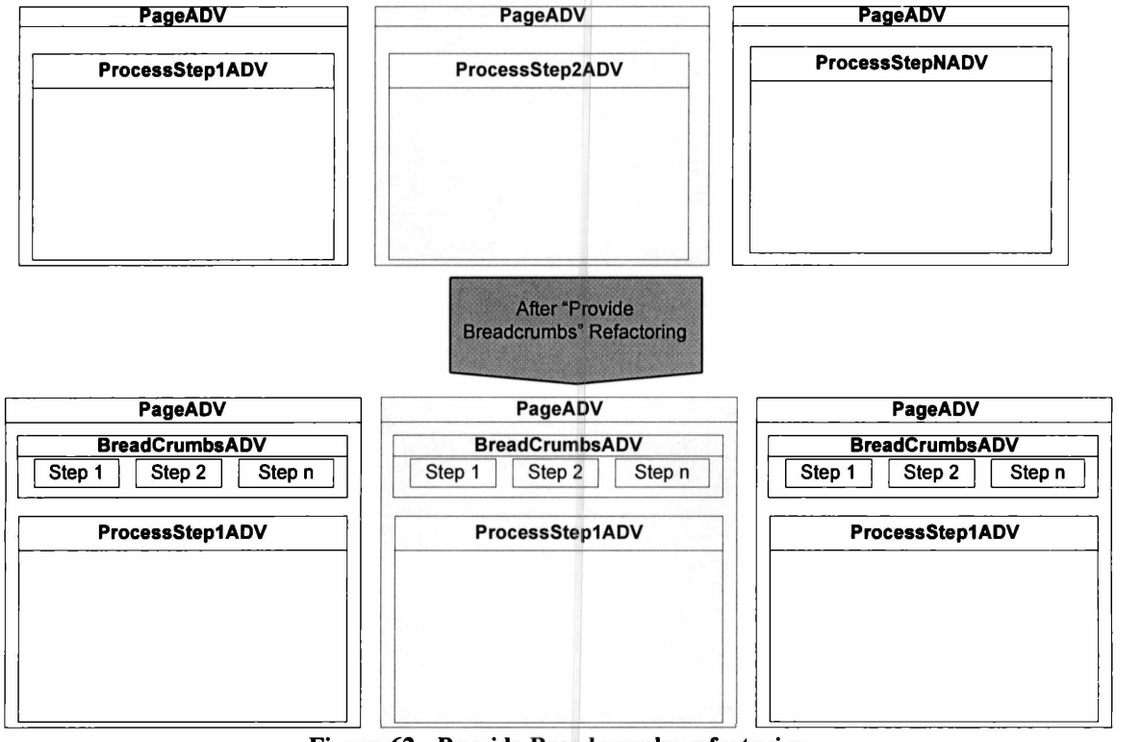


Figura 62 - Provide Breadcrumbs refactoring

4.10.3 Ejemplo

El sitio musimundo.com es un claro ejemplo donde se puede aplicar este refactoring. Cuando procedemos a realizar el checkout de un producto, nos encontramos con no tener ninguna información sobre en qué paso del proceso nos encontramos, cuanto falta para terminar, ni nos da ninguna alternativa de poder volver atrás en caso de cometer un error en el ingreso de los datos o cambiar de opinión. En las Figura 63 y Figura 64 se pueden ver dos screenshots del proceso de checkout, y en las Figura 66 y Figura 65 se muestra dos sitios que ya poseen esta funcionalidad y lo aplican de maneras distintas.

Figura 63 - Musimundo.com - proceso de checkout

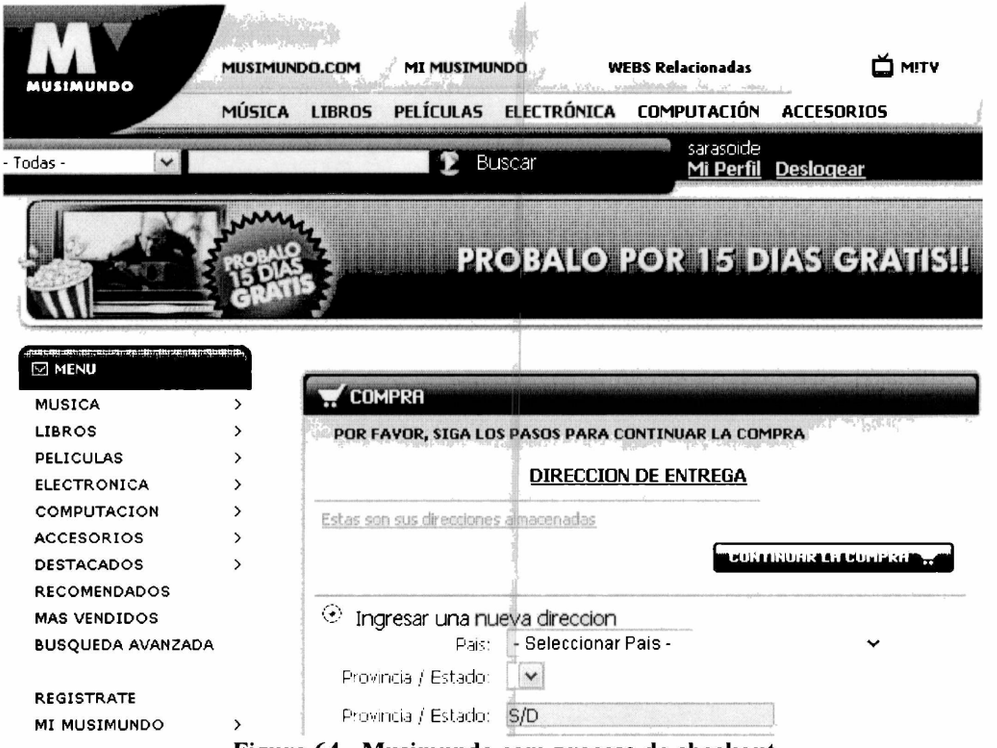


Figura 64 - Musimundo.com proceso de checkout

Figura 65 - Amazon.com - proceso de checkout

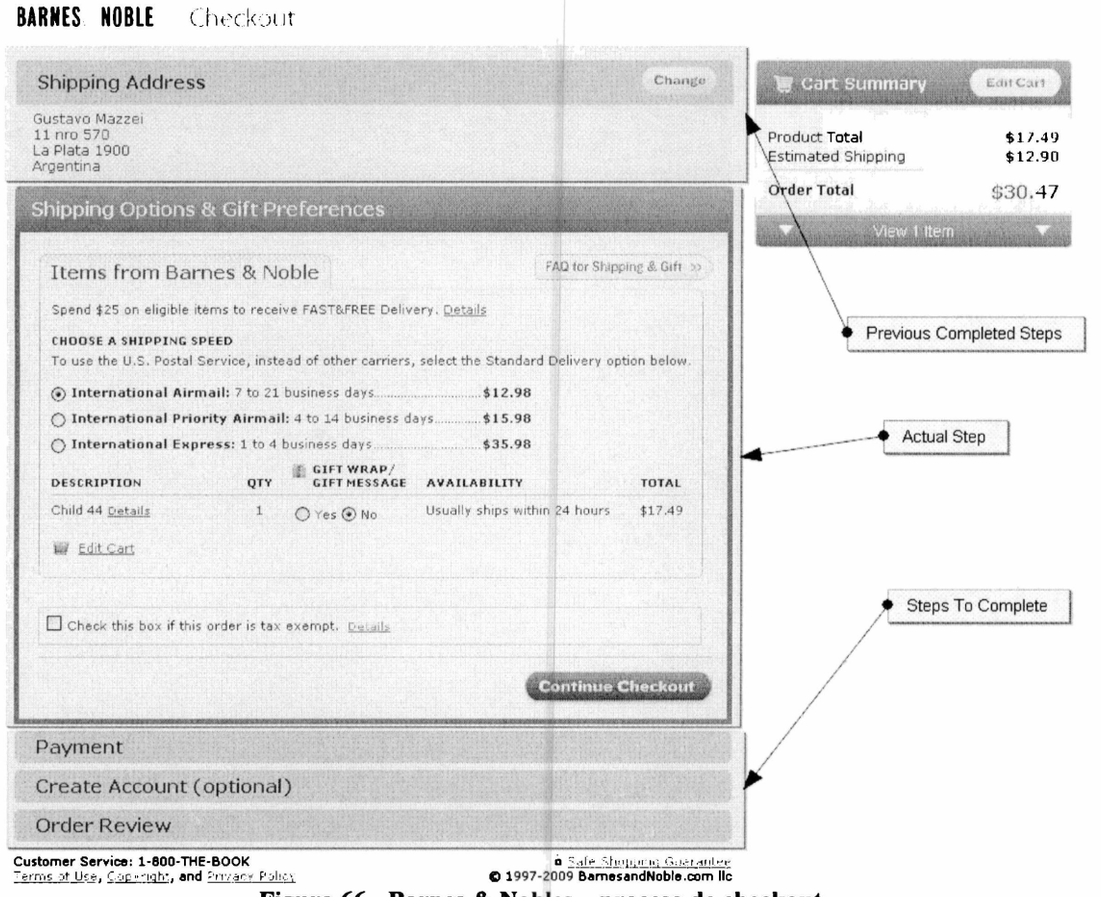


Figura 66 - Barnes & Nobles – proceso de checkout

4.11 Introduce Information on Demand

4.11.1 Motivación

Organizar la información relacionada de manera tal que el usuario no se sienta sobrecargado de información y tenga que hacer scrolling para encontrar la sección que busca.

4.11.2 Mecanismo

Primero debemos localizar el ADV que será objeto del refactoring; luego debemos verificar que la información que se quiere someter al refactoring ya se encuentra organizada en diferentes secciones, sino se debe aplicar el refactoring "Add Page Section".

Una vez que tenemos la información separada en secciones, procedemos aplicar comportamientos a los ADVs de las subsecciones. Para esto, definimos un template InfoOnDemandADV que instanciaremos con los ADVs de las subsecciones.

En la Figura 67 se puede ver el template del ADV y en la Figura 68 el ADV-chart del ADV.

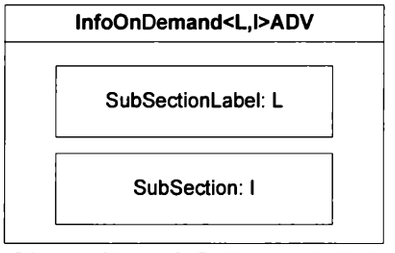
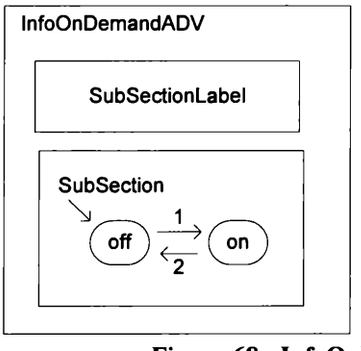


Figura 67 – InfoOnDemandADV



1:
Event: MouseClick
Pre-Cond: Focus(self)
Post-Cond: PerCont= PerCont+SubSection

2:
Event: MouseClick
Pre-Cond: Not Focus(self)
Post-Cond: PerCont = PerCont – SubSection

Figura 68 - InfoOnDemandADV – ADV-Chart

En la Figura 69 se puede ver el antes y el después de aplicar el refactoring en el modelo de presentación.

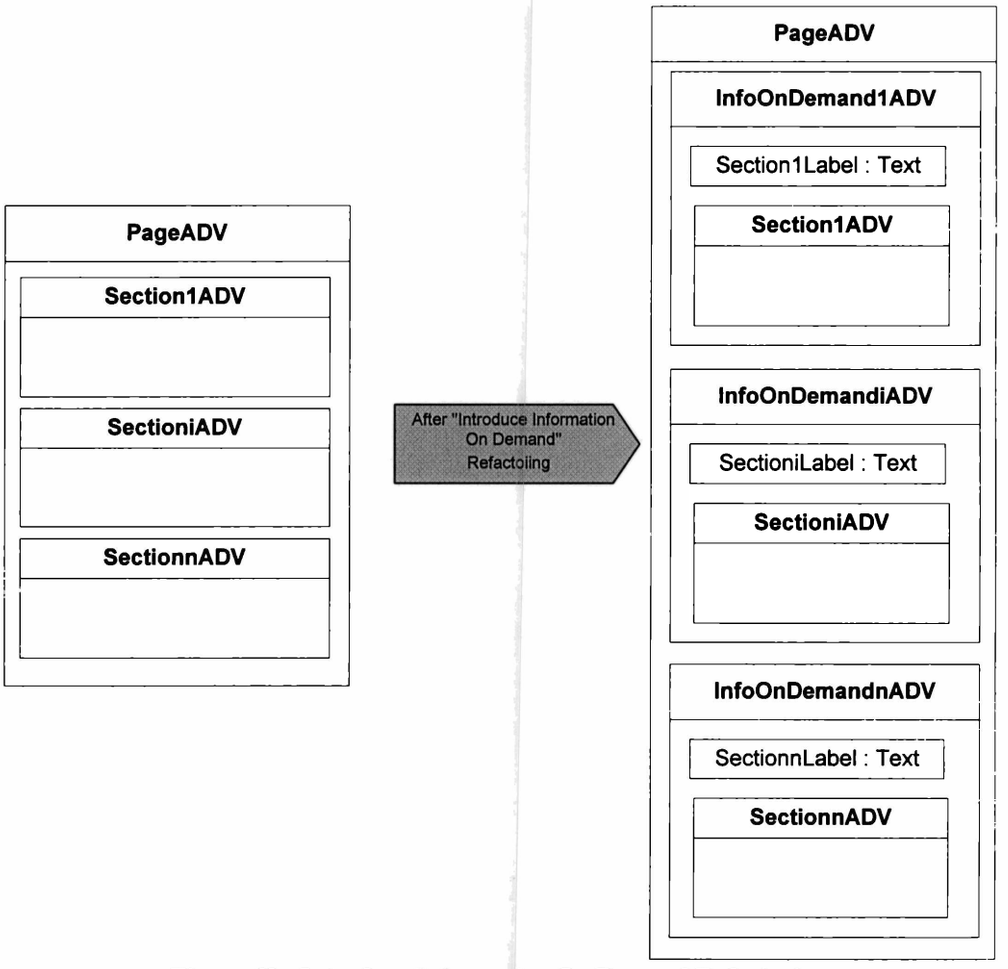


Figura 69 - Introduce Information On Demand Refactoring

4.11.3 Ejemplo

El sitio www.amazon.com que se caracteriza por poseer generalmente abundante información sobre el producto que se busca. Luego de ver los datos más importantes, como el precio, la imagen del producto, etc., se muestra información relacionada, como por ejemplo: revisiones del producto, otros productos que la gente compró después de comprar el producto, otros productos que la gente visitó cuando visitó el producto, etc.

Toda esta cantidad de información hace la página muy larga y por ende el usuario tiene que hacer mucho scrolling para llegar a la sección que le interesa. Esto se puede ver en Figura 70 y Figura 71.

Luego de aplicar este refactoring a esta página, podríamos tener pestañas (Tabs) que nos permitan mostrar la información a medida que se solicita sin necesidad de pasar por las que no nos interesan. Esto lo hace el sitio de Barnes & Nobles, ver Figura 72, cuando el usuario clickea sobre la pestaña la información relevante se muestra al usuario, esto permite al usuario tener una idea más clara del producto y no se siente sobrecargado de información.

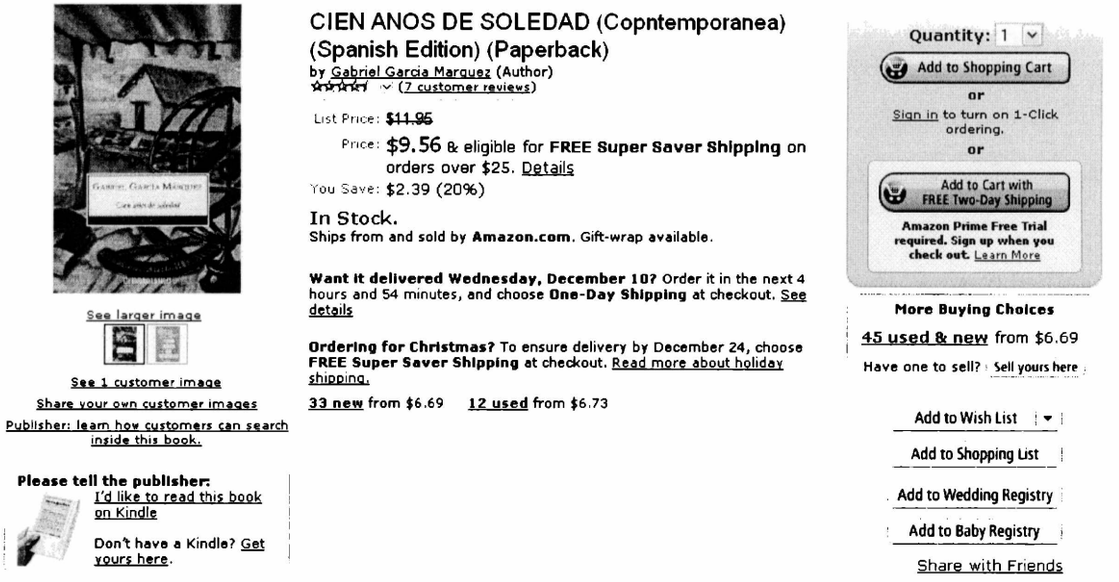


Figura 70 - Información de un producto en Amazon.com (1)

Special Offers and Product Promotions

- Haga sus compras en nuestra tienda de **Libros en Español** para obtener las últimas ediciones en lenguaje Español.

Frequently Bought Together

Total List Price: ~~$38.85~~
Price For All Three: $29.29
Add all three to Cart

- ☑ **This item:** CIEN ANOS DE SOLEDAD (Copntemporanea) (Spanish Edition) by Gabriel Garcia Marquez
- ☑ El amor en los tiempos del cólera (Oprah #59) by Gabriel Garcia Marquez
- ☑ Cronica de una muerte anunciada (Spanish Edition) by Gabriel Garcia Marquez

Customers Who Bought This Item Also Bought

Page 1 of 25

Cronica de una muerte anunciada (Spanish Edition) by Gabriel Garcia Marquez
★★★★½ (8) $9.56

Memoria de mis putas tristes (Spanish Edition) by Gabriel Garcia Marquez
★★★★ (53) $9.56

Del Amor Y Otros Demonios by Gabriel Garcia Marquez
★★★★½ (18) $8.95

Como agua para chocolate by Laura Esquivel
★★★★ (48) $11.16

Product Details

Paperback: 496 pages
Publisher: Plaza y Janes (February 7, 2006)
Language: Spanish
ISBN-10: 0307350428
ISBN-13: 978-0307350428

Figura 71 - Información de un producto en Amazon.com (2)

Cien años de soledad (One Hundred Years of Solitude)
by Gabriel García Márquez

Reader Rating: (22 ratings)
Detailed Rating:
"Dramatic" See All
› Read customer reviews **Write a Review**

Publisher: RH Espanol
Pub. Date: April 2006
ISBN-13: 9780307350428
Sales Rank: 7,465
496pp
Series: Contemporanea Ser.
Edition Description: Spanish-language Edition

Other Formats:
Paperback · Sp… e Edition

$10.75 Online price
(Save 10%)
$9.67 Member price
Join Now

Add to Cart

Add To List

4 used from …
Used Copies Available from our Authorized Sellers.

FAST FREE … Details

Holiday Delivery Schedule
Delivery Times and Shipping Rates

Enter a zip code

Tabs

Table of Contents

Overview | Editorial Reviews | Customer Reviews | Meet the Writer | Features | Full Product Details

Synopsis
Cien años de soledad es una obra maestra del escritor colombiano Gabriel García Márquez, quien obtuvo el Premio Nobel de Literatura en 1982. El libro sigue la historia de seis generaciones de la familia Buendía y cien años de vida en el ficticio pueblo de Macondo, mariposas relatando sus pesares, problemas, desventuras y esperanzas, en su esfuerzo por vivir en tan emblemático pueblo. La historia tiene un final místico e inesperado, que cierra todos los cabos que el autor va dejando abierto durante el relato.

› More Reviews and Recommendations

A chief practitioner of the "magic-realist" style, Gabriel García Márquez's influence and importance lie in his crucial role of bringing Latin-American fiction to wider audiences while pioneering it at the same time. The Colombian-born Nobel winner

Reader Rating:
Ratings: 22 Reviews: 17
› See All Reviews

WRITE A REVIEW

Cien Años de Soledad
by FocoProject

Tabs

Customers who bought this also bought
More by This Author
Related Authors
Related EssentiaLists
Fans of this Book
Related Subjects
Search Related Categories
Of Interest
Order Profile of a Writer: Marquez - Tales Beyond

Figura 72 – Información de un producto de Barnes & Nobles.

4.12 Introduce Link Destination Announcement

4.12.1 Motivación

La motivación de este refactorings es enriquecer el ancla de un link con widgets adicionales para presentar información sobre la página destino.

Algunas interfaces gráficas presentan una lista larga de elementos al usuario, por ejemplo como resultado de una operación de búsqueda. Tener que navegar a cada elemento de la lista y luego volver al resultado de la búsqueda puede ser tedioso para el usuario. Este refactoring propone dar un preview con algunas posibles operaciones sobre el elemento cada vez que se posiciona sobre el link.

4.12.2 Mecanismo

Este refactoring se realiza en dos pasos:

1. El primer paso para este refactoring involucra hacer un objeto de la interfaz sensible al evento "MouseOver".

2. Luego, asociar el evento del MouseOver a la acción de mostrar algunos objetos de interfaz que se correspondan con el link de destino. Estos objetos aparecerán y desaparecerán cuando el usuario se posicione el mouse sobre el link.

En la Figura 73 y la Figura 74 se muestra el template del ADV para este refactoring que se usará para mostrar los detalles del destino target.

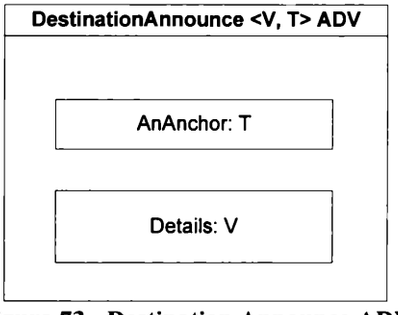


Figura 73 - Destination Announce ADV

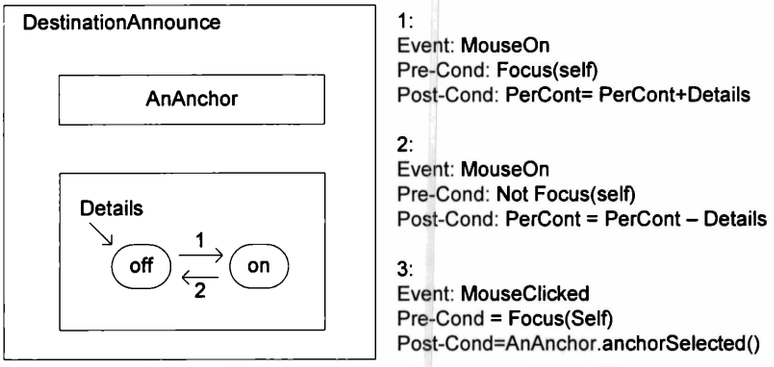


Figura 74 - Destination Announce ADV-Chart

Por ejemplo, si se tiene una lista de productos, se creará un ProductADV con la información que deseamos mostrar de cada producto y se creará una instancia de DestinationAnnouceADV, donde V es ProductADV, que aparecerá cada vez que posicione sobre el link de cada resultado.

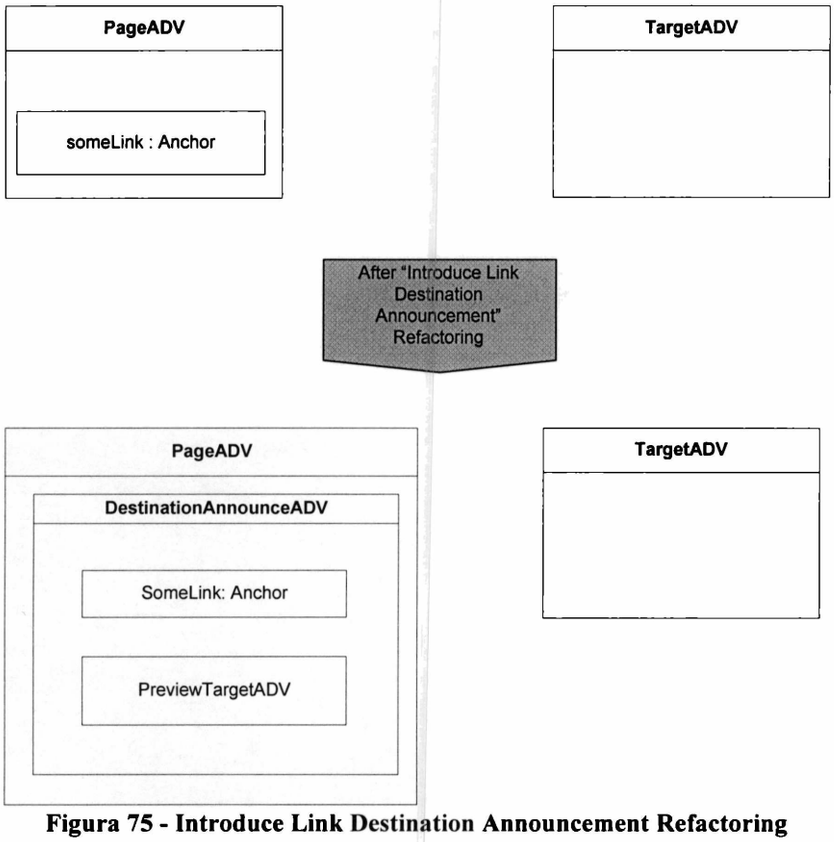


Figura 75 - Introduce Link Destination Announcement Refactoring

4.12.3 Ejemplo

En el sitio www.amazon.com, podemos observar que luego de buscar un artículo siempre se muestran recomendaciones basadas en lo que otros usuarios también compraron luego de comprar el producto que se está visitando.

En la Figura 76 podemos ver que luego de buscar el libro "100 años de soledad" nos encontramos con una recomendación del libro "Crónica de una muerte anunciada" del mismo autor, para ver los detalles o para agregar al carrito de compras de esa recomendación debemos navegar a la página de detalles, que se ve en la Figura 77, y luego hacer click en agregar al carro de compras

En la Figura 78 podemos ver como quedaría la página original luego del refactoring y como se facilita al usuario el proceso de compra sin necesidad de navegar a la página de los detalles de la recomendación.

Figura 76 - Recomendación del sitio amazon.com al navegar el ítem "cien años de soledad"

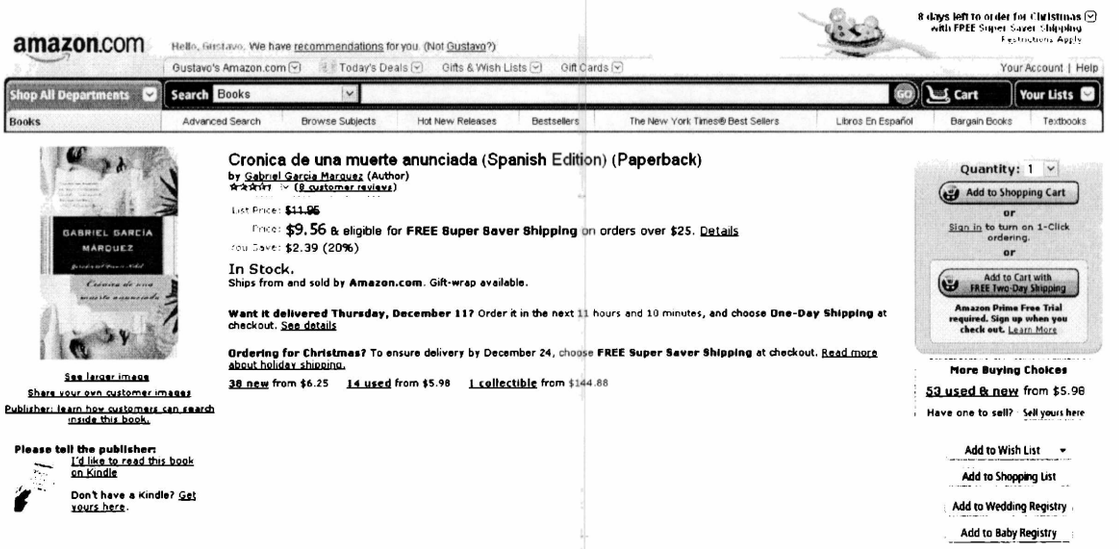


Figura 77 - Luego de clickear en "crónica de una muerte anunciada"

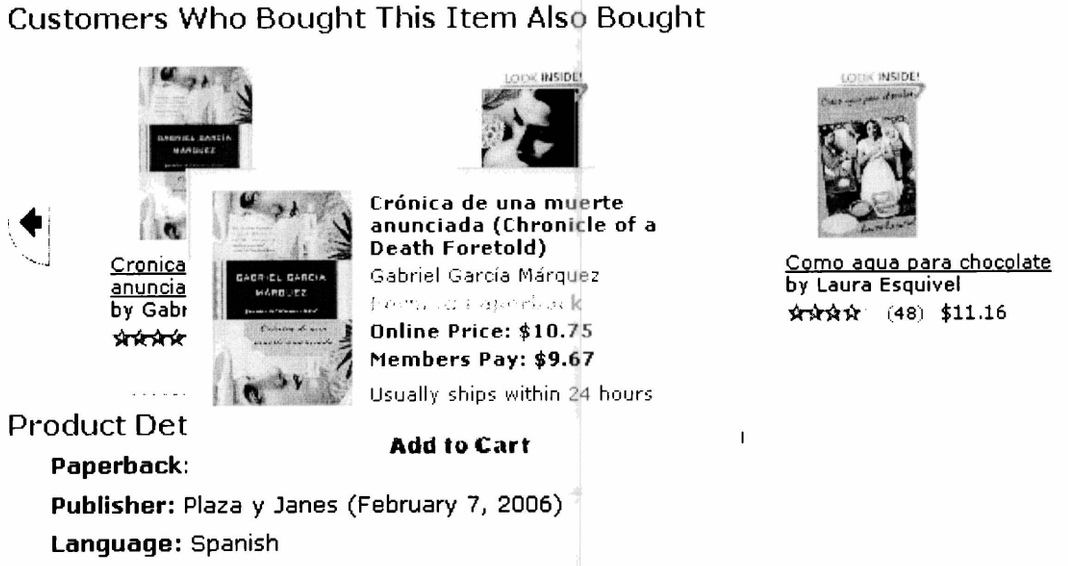


Figura 78 - Sitio luego del refactoring "Introduce Link Destination Annoucement"

4.13 Introduce Scrolling

4.13.1 Motivación

Permite arreglar los problemas de tamaño de un widget de presentación mientras se muestra gran cantidad de información de texto u otra información en el mismo.

4.13.2 Mecanismo

Definimos un template ScrollableIndexADV que está compuesto por dos botones que representar ir hacia delante y atrás (arriba o abajo) y uno o más ADV que representen el ítem que se está browseando. En la Figura 79 se muestra el template del ADV, y en la Figura 80 se muestra el ADV-chart del mismo.

El procedimiento consiste en reemplazar los ADVs que se mostraban en la página a aplicar el refactoring por el nuevo ScrollableIndexADV instanciado con el ADV del item que se va a mostrar. El procedimiento se muestra en la Figura 81.

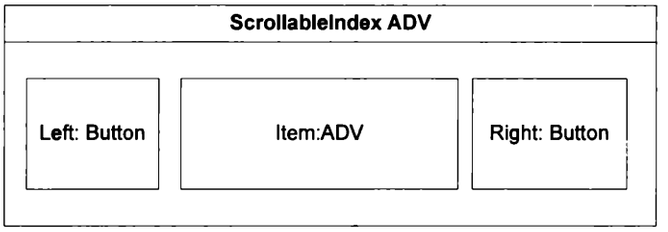
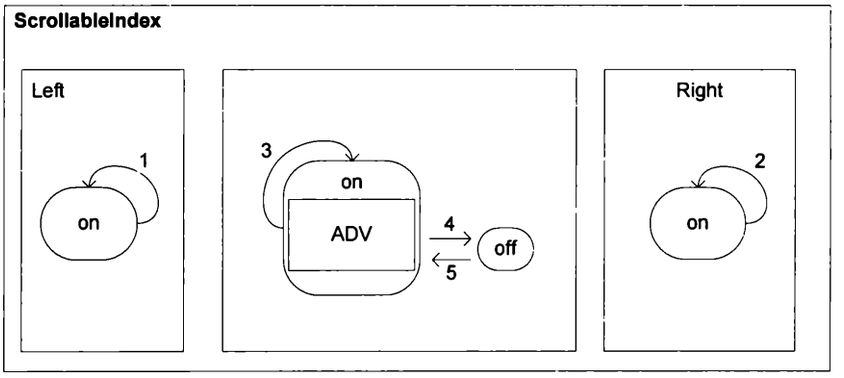


Figura 79 - ScrollableIndex ADV



1:
Event: MouseClicked
Pre-Cond: Focus(Left)
Post-Cond: ShiftLeft()

2:
Event: MouseClicked
Pre-Cond: Focus(Right)
Post-Cond: ShiftRight

3:
Event: MouseClicked
Pre-Cond: Focus(items[i])
Post-Cond: items[i].anchorSelected()

4,5:
Event: ShiftLeft
Pre-Cond: OFFSET>0;
((V i) (0<=i < 5 && items.get(i).getModel() == owner.items.get(OFFSET+i))
Post-Cond: (V i) (0<= i<5);
items.get(i).getModel() == owner.items.get(OFFSET +i -1)

Event: ShiftRight
Pre-Cond: OFFSET<owner.Items.size()-5
((V i) (0<= i < 5 && Items.get(i).getModel() == owner.Items.get(i+OFFSET))
Post-Cond: (V i) (0<= i < 5)
Items.get(i).getModel()==owner.Items.get(OFFSET +i + 1))

Figura 80 - ScrollabeIndex ADV – Chart

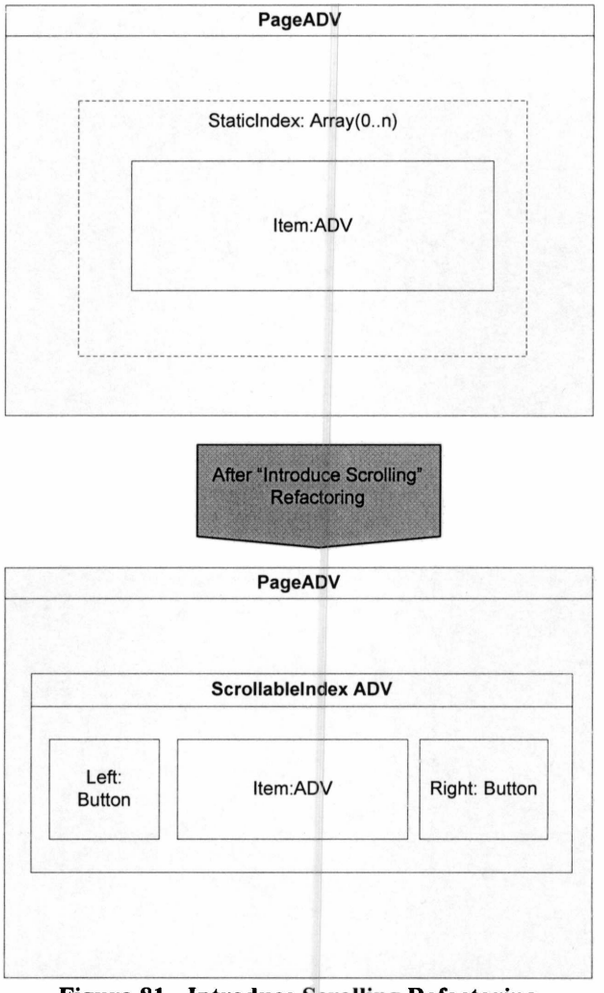


Figura 81 - Introduce Scrolling Refactoring

4.13.3 Ejemplo

En el sitio temática.com en este caso se buscó un libro popular, como "100 años de soledad" y se ve que la empresa decidió mostrar también recomendaciones para la gente que compró el libro de acuerdo al autor y obras similares, pero las recomendaciones se limitan en tamaño debido a que la página se haría muy extensa, en este caso podemos aplicar el refactoring para utilizar el mismo espacio en la página y permitir al usuario ver mayor cantidad de recomendaciones.

En la Figura 82 se muestra el sitio y recuadrado en verde las secciones que serán afectadas por el refactoring.

SUBCATEGORIAS DE LIBROS

Arte, Arquitectura y diseño (2902)
Autoayuda (3774)
Ciencias de la salud, Naturales y divulgacion cientifica (3104)
Computación y sistemas (560)
Derecho y ciencias sociales (4709)
Enciclopedias, Diccionarios, Linguística (638)
Esoterismo (1928)
Ficción y literatura (10693)
Gastronomia y costumbres (1207)
Hogar (1313)
Hoteleria y turismo (1347)
Humanidades (8451)
Infantil y juvenil (8501)
Ingenieria, Tecnica y ciencias exactas (805)
Juegos, Deportes y recreacion (1329)
Negocios y cs. Economicas (2307)
No catalogados (476)
Textos (540)
Visuales (501)

Eventos
Asociados web
Referidos
Servicios al cliente

CIEN AÑOS DE SOLEDAD
GABRIEL GARCIA MARQUEZ
ISBN: 9788420471839
Editorial: Alfaguara
Clasificación: Ficción y Literatura
Páginas: 756
Publicación: Marzo 2007 | Idioma: Español
Formato: Tapa Dura

1 Comentario

*Salida habitual del deposito en 72 horas

AMPLIAR IMAGEN

PRECIO $ 39,00.- | U$S 11,38.- | € 9,00.- COMPRAR

Ver plan de cuotas

COMPARTIR CON MIS REDES SOCIALES Y AMIGOS

AGREGAR A MI LISTA DE DESEOS

e xtra

COMPRANDO ESTE PRODUCTO PODES SUMAR:
117 PUNTOS eXtra!
Ingresar a eXtra!

GASTOS DE ENVIO DE ESTE PRODUCTO, A PARTIR DE:
$ 12,10.-
Ver costos de envio y Tiempos de Entrega

Envío Gratis en Argentina con tus compras mayores a $150.- No Acumulable con otras promociones.

RESEÑA

Cien Años de Soledad - Gabriel Garcia Marquez

Sinopsis
En Cien años de soledad aparece ante nuestros ojos todo un mundo. Para preparar el acercamiento a él, abren la edición una breve semblanza de García Márquez escrita por Álvaro Mutis y una introducción de Carlos Fuentes que aporta testimonios personales esclarecedores sobre el nacimiento de la novela y el deslumbramiento inmediato que suscitó. El magistral análisis que Mario Vargas Llosa hizo de la narrativa de García Márquez sigue siendo la más alta referencia y de él se ofrece una parte central. Dos estudios, de Víctor García de la Concha y de Claudio Guillén -a quien la muerte sorprendió cuando le ponía punto final-, tratan de mostrar caminos concretos de acceso al texto de la novela.

Tras él, cuatro acade[illegible] Barcia (Argentina), Juan Gustavo Cobo [illegible]éxico) y Sergio Ramírez (Nicaragua)- glosan distintos aspectos de la personalidad literaria de García Márquez y de lo que Cien años de soledad significó en la trayectoria de la novela hispánica y en su amplia difusión en el ámbito cultural.

La Asociación de Academias de la Lengua presentará oficialmente su edición, en el marco del IV Congreso Internacional de la Lengua Española, en el homenaje a Gabriel García Márquez que se celebrará el 26 de marzo y en el que se entregará al autor el primer ejemplar.

ACERCA DE ESTA OBRA

VER PRIMER CAPITULO VER INDICE DE CONTENIDOS

VER BIOGRAFIA DEL AUTOR VER ENTREVISTAS AL AUTOR

COMENTARIOS

Adrián Elbio Lugano dijo:

Éste libro tiene que ser leido por todos.La basta imaginación de Gabriel García Marquez, para cear personajes, situaciones e historias, narradas con un sentido del humor único, transforman a éste libro en una pieza única e imperdible.

Calificación: Excelente

AGREGAR COMENTARIO

Ver mas productos de:
Ficción Y Literatura

LOS QUE COMPRARON OBRAS DE GABRIEL GARCIA MARQUEZ TAMBIEN COMPRARON:

LA FORET DES PYGMEES
Isabel Allende

ZORRO
Isabel Allende

LA FIGLIA DELLA FORTUNA
Isabel Allende

AFRODITA
Isabel Allende

ALEPH, L'
Jorge Luis Borges

LOS QUE COMPRARON CIEN AÑOS DE SOLEDAD TAMBIEN COMPRARON:

ATROZ ENCANTO DE SER ARGENTINOS 2
MARCOS AGUINIS

LA SUMA DE LOS DIAS
ISABEL ALLENDE

MUERTOS DE AMOR
JORGE LANATA

LA VIDA ETERNA
FERNANDO SAVATER

LAS PEQUEÑAS MEMORIAS
JOSE SARAMAGO

Figura 82 - Sitio tematika.com

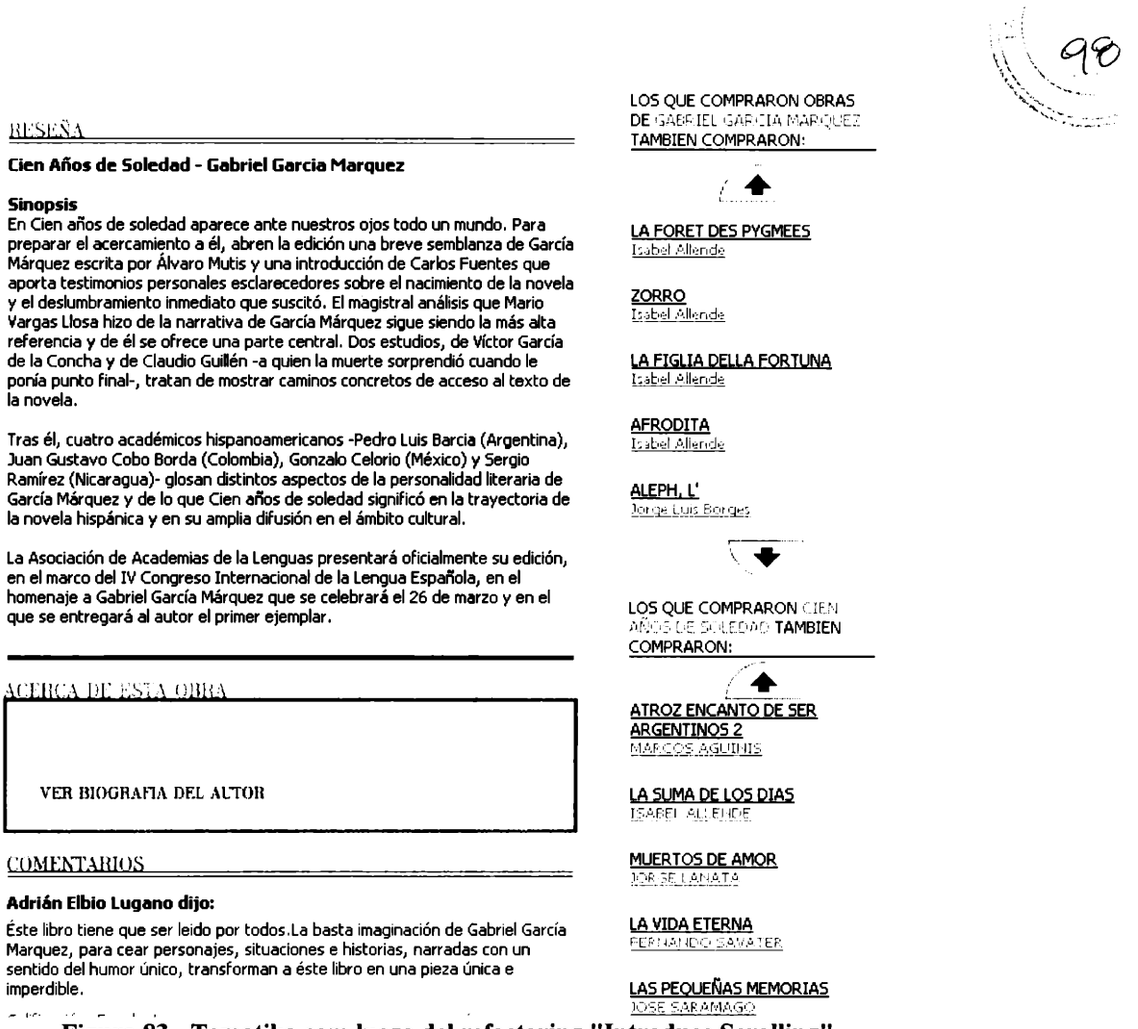


Figura 83 - Tematika.com luego del refactoring "Introduce Scrolling"

4.15 Introduce Pagination

4.15.1 Motivación

Similar a "Split Page" nos permite dividir una lista de ítems que creció en tamaño en más listas, esto generalmente se conoce como paginación.

Este refactoring es muy común y está presenta en casi todas las aplicaciones actuales.

4.15.2 Mecanismo

Previamente a introducir este cambio, el modelo de dominio de la aplicación debe permitir la paginación, y el modelo de navegación debe poseer una forma de brindarnos la información de los nodos por páginas.

Una vez que se tiene esta funcionalidad en el modelo de navegación, se debe modificar el ADV que se encuentra originalmente, y agregar el ADV que nos permita solicitar la página siguiente y la anterior, o generar widgets con los números de página.

En la Figura 84 se puede ver un esquema del procedimiento.

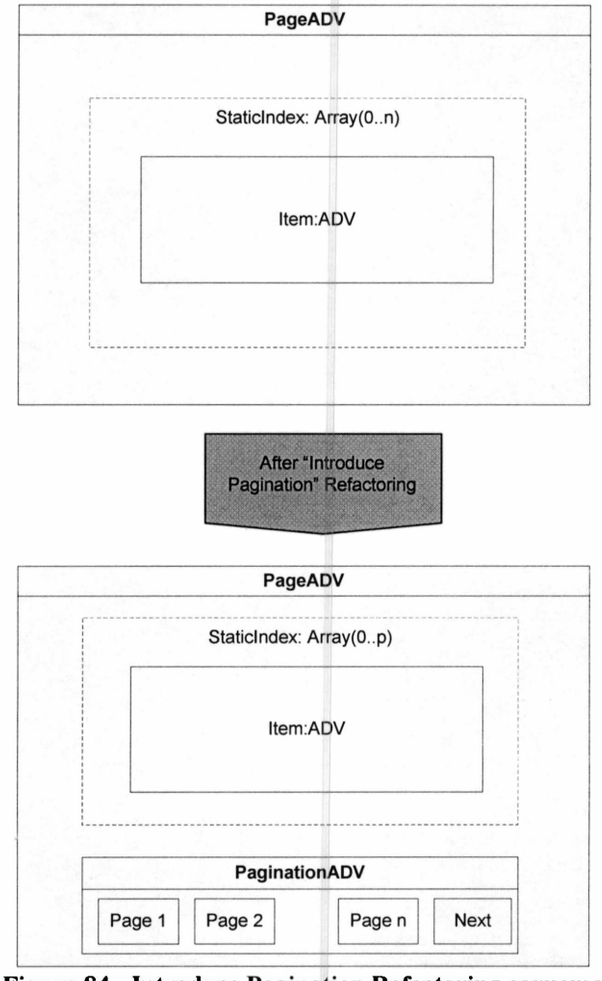


Figura 84 - Introduce Pagination Refactoring esquema

4.15.3 Ejemplo

En el sitio de Facebook, el news-feed de un usuario puede crecer indeterminadamente si se poseen muchos contactos y estos realizan muchas acciones, aunque Facebook nos permite paginar alguna de esta información, generalmente lo hace cuando la lista es casi interminable.

En la Figura 85 se puede ver un ejemplo de la lista de news de un perfil de Facebook y en su derecha se muestra como quedaría luego del refactoring.

Figura 85 - Introduce Pagination Refactoring

Capítulo 5
Conclusiones

Esta tesis presenta un conjunto de mejoras para las aplicaciones Web que se aplican a sus modelos de navegación y presentación de acuerdo a la metodología OOHDM. Esto representa una contribución a la definición de refactorings para aplicaciones Web.

La principal conclusión que se puede sacar de este trabajo es que es posible mejorar de manera incremental y simple una aplicación Web mediante pequeños cambios y mejoras en sus modelos de navegación y presentación, sin cambiar los requerimientos básicos de la aplicación.

Esta tesis provee un catalogo bien documentado de refactorings para el modelo de navegación de aplicaciones Web, mediante ejemplos resueltos sobre diagramas de navegación de OOHDM, mostrando su uso en aplicaciones del mundo real.

También provee un catalogo bien documentado de refactorings para el modelo de presentación de aplicaciones Web, mediante ejemplos del mundo real utilizando diagramas de presentación de OOHDM y mostrando su uso en aplicaciones muy conocidas de Internet.

Con la expansión de las aplicaciones Web y la evolución de las nuevas tecnologías (la denominada Web 2.0 o aplicaciones RIA), nuevos refactorings irán surgiendo que amplíen este catálogo.

5.1 Trabajos Futuros

Dado que la evolución de las aplicaciones Web en la actualidad es vertiginosa, la aparición de nuevos patrones Web es inevitable. En el sitio [21] podemos ver un catálogo muy completo de estos patrones que se actualizan continuamente a medida que se descubren nuevos patrones. En el presente trabajo sólo se utilizaron algunos de los patrones Web como motivación de algunos refactorings presentados. Por lo tanto, los nuevos patrones o algunos de los ya existentes pueden surgir como motivación de nuevos refactorings en los distintos modelos de navegación y presentación de aplicaciones Web.

También en este trabajo mostramos algunos refactorings que son aplicables a aplicaciones RIA; una futura extensión a este catálogo podría incluir refactorings exclusivos para este tipo de aplicaciones.

Esta tesis presenta bases para futuros desarrollos en el campo del refactoring de modelos de presentación y navegación de una aplicación Web.

Como desarrollo futuro derivable de este trabajo podría ser la construcción de herramientas que permitan la automatización de la técnica de refactoring aplicando las transformaciones mencionadas en los modelos de navegación y presentación.

El alcance de esta tesis se limitó a mostrar la técnica del refactoring hasta el modelo de presentación definido por OOHDM, pero no muestra como estos refactorings serían aplicados sobre alguna implementación Web popular como GWT, Struts, etc.
Una futura extensión que se puede derivar de este trabajo es mostrar la técnica de refactoring, en alguna implementación particular usando frameworks de aplicaciones Web.

Dado que esta tesis no incluye en los refactorings de modelo de navegación a los diagramas de contexto, un futura extensión podría ser ampliar el catalogo de refactorings de modelo de navegación aquí mostrado, para incluir los refactorings que se apliquen sobre los diagramas de contexto de navegación.

Bibliografía

[1] N. Koch and A. Kraus. The expressive power of UML-based web engineering. In Proc. of 2nd Int. Workshop on Web Oriented Software Technology (IWWOST02) at ECOOP02, pages 105{119, Malaga, Spain, 2002.

[2] UWA Consortium. Ubiquitous web applications. In Proceedings of the eBusiness and eWork Conference e2002, Prague, Czech Republic, 2002.

[3] D. Schwabe and G. Rossi. An object oriented approach to web-based application design. Theory and Practice of Object Systems, 4(4), 1998. Wiley and Sons.

[4] Marko Boger, Thorsten Sturm, and Per Fragemann. Refactoring browser for UML. In Objects, Components, Architectures, Services, and Applications for a NetworkedWorld: International Conference NetObjectDays, NODe 2002, Erfurt, Germany, volume LNCS 2591/2003, pages 366{377. Springer Berlin, 2003.

[5] Beck, Kent. Extreme Programming explained: Embracing Change. Reading, Mass., Addison-Wesley, 1999.

[6] F. Ricca and P. Tonella. Program transformations for web application restructuring. In Web Engineering: Principles and Techniques. Woojong Suh (eds.), chapter XI, pages 242-260. 2005.

[7] Fowler, M. Refactoring: Improving the Design of Existing code. Addison-Wesley, 2000.

[8] T. Mens and T. Tourwé, "A Survey of Software Refactoring," IEEE Transactions on Software Engineering, vol. 30, pp. 126-239, 2004.

[9] T. Mens, S. Demeyer, and D. Janssens, "Formalizing behaviour preserving program transformation," presented at 1st Int. Conf. on Graph Transformation, 2002.

[10] W. F. Opdyke, "Refactoring: A Program Restructuring Aid in Designing Object-Oriented Applicatin Frameworks," University of Illinois, 1992.

[11] J. Overbey, S. Xanthos, R. Johnson, and B. Foote. Refactorings for Fortran and High-Performance Computing. In 2nd. Int. Workshop on Software Engineering for High Performance Computing System Applications, St. Louis, MO, 2005.

[12] Alejandra Garrido. Program Refactoring in the Presence of Preprocessor Directives. PhD thesis, University of Illinois at Urbana-Champaign. Tech.Rep.No. UIUCDCS-R-2005-2617, October 2005.

[13] Antonio Menezes Leit~ao. A formal pattern language for refactoring of Lisp programs. In CSMR'02: Proceedings of the Sixth European Conference on Software Maintenance and Reengineering, page 186, Washington, DC, USA, 2002. IEEE Computer Society.

[14] Filippo Ricca, Paolo Tonella, and Ira D. Baxter. Restructuring web applications via transformation rules. In SCAM 2001, pages 150{160, 2001.

[15] UML® Resource Page, OMG, http://www.uml.org/

[16] S. Ceri, P. Fraternali, and A. Bongio. Web modeling language (WebML): a modeling language for designing web sites. In Proc WWW9 Conference.(also in Computer Networks, 33 (2000), Amsterdam, NL, 2000.

[17] Jordi Cabot and Cristina Gomez. A catalogue of refactorings for navigation models. In Proc. Of the 8th. Int. Conference on Web Engineering: ICWE'08, Yorktown Heights, New York, 2008.

[18] Garrido, G. Rossi, and D. Distante, "Model Refactoring in Web Applications," presented at 9th Int. Symp. on Web Site Evolution (WSE'07), 2007.

[19] D.Van Duyne, J. Landay, and J. Hong. The Design of Sites. Addison-Wesley, 2003.

[20] Web Design Patterns. http://www.welie.com/ patterns/.

[21] Web Patterns. http://webpatterns.org.

[22] Gustavo Rossi, Mario Matias Urbieta, Jeronimo Ginzburg, Damiano Distante, and Alejandra Garrido. Refactoring to rich internet applications. a model-driven approach. In Proc. of the 8th. Int. Conference on Web Engineering: ICWE'08, Yorktown Heights, New York, 2008.

[23] Gustavo Rossi, Daniel Schwabe, and Alejandra Garrido, Design reuse in hypermedia applications development. In proceedings of Hypertext '97. Southampton, UK, 1997

[24] A. Garrido, G. Rossi y D. Distante. Systematic Improvement of Web Application Design. To appear in Journal of Web Engineering, 2009.